UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

TESIS DOCTORAL

TÉCNICAS DE PARTICIONAMIENTO MULTIDIMENSIONAL BASADAS EN ÍNDICES MULTIATRIBUTO EN BASES DE DATOS PARALELAS

MANUEL BARRENA GARCÍA

Madrid, Noviembre de 1995

A Juan y Adela,
a quienes todo les debo

Agradecimientos

Afortunadamente, son muchas las personas a las que debo agradecer el que este proyecto haya podido ver la luz sin perder en el camino ni un gramo de la ilusión con el que lo inicié.

En primer lugar, quiero agradecer a Pedro de Miguel, director de esta tesis, a enseñarme a distinguir el grano de la paja. Sus juicios, bien dirigidos y siempre atinados, me ayudaron a comprender en gran medida el valor de las cosas bien hechas. Mi gratitud también por brindarnos su apoyo y ofrecernos su confianza cuando la situación era difícil. Él es el principal responsable de la evolución positiva que ha sufrido el grupo QUATRO de Cáceres a lo largo de estos años.

Gracias también a Manolo Nieto, sin cuya paciencia para escuchar detenidamente nuestras ideas, su disposición constante para solucionarnos el papeleo que acarrea todo el proceso y, en definitiva, su apoyo permanente para resolvernos las dudas técnicas que le hemos ido planteando, hubiera sido mucho más lento y difícil conseguir nuestros objetivos.

Mi gratitud asímismo a todo el personal del Departamento de Arquitectura y Tecnología de Sistemas Informáticos de la U.P.M., por habernos hecho tan agradable nuestras visitas a la Facultad durante la fase de elaboración de esta tesis.

Quiero destacar también la enorme suerte que he tenido de pertenecer al grupo QUATRO. Todos los componentes de este grupo merecen mi cariño y con él, este sentimiento de gratitud enorme por haberme sentido feliz trabajando con ellos. Menciono en primer lugar a Juan Hernández, mi compañero de despacho, con quien he compartido más de cerca las alegrías por ver cumplir los objetivos propuestos, así como alguna que otra decepción por no conseguir los resultados esperados. A Juan le agradezco la posibilidad que me ha dado de disfrutar de una relación laboral enriquecedora en la que, desde siempre, éxitos y fracasos han sido compartidos sin falsas hipocresías. Sus palabras de aliento en horas bajas y su confianza en mi trabajo han supuesto un importante resorte para terminar con éxito este trabajo. Mi gratitud a Jose Martínez Candela, promotor inicial de nuestro grupo y bandera permanente del mismo. Su sentido crítico del trabajo me ayudó a pulir y mejorar muchos de los informes que pasaron por su mano. A Antonio Polo, tercero de los componentes del grupo QUATRO y compañero permanente de tareas docentes, también le doy las gracias por despejarme tantas dudas y sacarme del atolladero en tantas ocasiones. Su visión rigurosa y su concepto ordenado del método científico, me ayudaron a plantear con seriedad muchos de los trabajos reflejados en esta tesis.

Debo extender mi gratitud a todos los compañeros del Departamento de Informática de la Universidad de Extremadura, que me animaron en todo momento a sacar adelante este proyecto y me permitieron acaparar, durante largo tiempo, la mejor impresora del departamento con el fin de imprimir esta memoria.

Agradezco también a la profesora Betty Salzberg y a Georgios Evangelidis, de la North-eastern University, en Boston, el interés que mostraron al hacerse eco de mis ideas sobre el árbol Q. Sus comentarios a mis trabajos y su permanente disposición para brindarme información actualizada sobre sus árboles hB, me allanaron mucho el camino para llegar a este punto.

No puedo dejar de mencionar a toda mi familia, a la que durante tanto tiempo he desatendido por realizar este trabajo. Mi gratitud a todos ellos por haber antepuesto su comprensión a su necesidad por tenerme con ellos. En este mismo sentido, agradezco también a Paco y Fernando, mis entrañables amigos, el cariño con el que siempre me han tratado y que no he sabido devolver en igual medida durante estos últimos años.

Mi último y más hondo sentimiento de gratitud se lo dedico a Montse, mi mujer, cuyo amor ha sido el sustento de todo mi trabajo. Su amor, generosidad e infinita paciencia han sido la clave para que este proyecto haya podido convertirse en realidad.

Resumen

Los requerimientos cada día más exigentes de modernas aplicaciones de bases de datos, tales como GIS, CAD, CASE y otras, imponen la necesidad de encontrar nuevas vías de solución al problema del tratamiento de grandes volúmenes de información. La potencia de procesamiento de computadores paralelos económicamente abordables, ha atraído la atención de una gran comunidad de investigadores y técnicos que encuentran en los sistemas paralelos de bases de datos la respuesta eficiente a las exigencias de nuevas aplicaciones.

Específicamente, la tecnología del paralelismo resulta una atractiva vía de solución a la problemática tradicional del cuello de botella que representan las operaciones de entrada/salida. Con objeto de minimizar el tiempo de respuesta a una consulta, los sistemas de bases de datos paralelas particionan los datos entre un conjunto de dispositivos de almacenamiento, favoreciendo el acceso en paralelo a los mismos y permitiendo, en definitiva la participación concurrente de varios procesadores en la ejecución de una consulta.

Habitualmente, el particionamiento de las relaciones se efectúa por un sólo atributo, enviando las tuplas a distintos dispositivos dependiendo del valor de dicha tupla sobre el atributo de particionamiento. Esta forma de fragmentar los datos resulta adecuada cuando el predicado de la consulta incluye el atributo de particionamiento. Sin embargo, en aquellos casos en que esto no sea así, la consulta debe ser dirigida hacia todos los nodos de procesamiento encargados de gestionar algún fragmento de la relación o relaciones implicadas en la consulta. Este modo de proceder afecta negativamente no sólo al tiempo de ejecución de la consulta, sino también al *throughput* del sistema.

En la tesis que se presenta, se proponen modelos de particionamiento multidimensional, basados en la consideración de múltiples atributos. Básicamente, la técnica propuesta consiste en realizar un particionamiento por múltiples dimensiones del espacio de tuplas, enviando posteriormente los diferentes fragmentos en que queda dividido este espacio a un determinado número de discos del sistema.

Por su parte, la fragmentación del espacio de tuplas se realiza equilibradamente por medio de un nuevo mecanismo de indexación multiatributo, conocido bajo el nombre de árbol Q.

En el desarrollo de esta memoria de tesis, se exponen las ideas que han conducido al establecimiento del árbol Q; se definen con detalle las estructuras y algoritmos de manipulación del árbol Q; se presentan diversas estrategias de particionamiento basadas en esta estructura y se exhiben los resultados de rendimiento de las diferentes propuestas, basados en los trabajos de implementación realizados durante la fase de ejecución de esta tesis.

Abstract

The demanding requirements of modern database applications, such as GIS, CAD, CASE and others, claim for new solutions to the problem of managing large quantity of information. The processing power of inexpensive parallel computers has focussed the attention of many searchers who find in such computer systems the answer to the demands of these new applications.

Specifically, the parallelism technology seems an attractive via to solve the traditional bottlelneck found in input/output operations. With the goal of minimizing the response time of a query, the parallel database systems decluster data among a number of storage devices, by favouring the access in parallel to data and by permitting the contribution of several processors in the execution of a query.

Frequently, the partitioning of relations is made by a single attribute, sending tuples to different disks by depending on the value of the tuple on the partitioning attribute. This way to fragment data is useful when the partitioning attribute is involved in the predicate of the query. However, in those situations where it is not the case, the query must be directed to every processing node which is in charge of some fragment of the relation or relations involved in the query. This approach affects negatively to both, the response time of the query and the throughput of the system.

In the thesis we present, a multidimensional partitioning model is proposed. In short, the proposed technique partitions, on the base of multiple attributes, the tuple space by sending the different fragments of the space to a specific number of disks in the system.

By its hand, the tuple space partitioning is made in a balanced way by means of a new multi-attribute indexing method, called the Q-tree.

In this thesis dissertation, we present the ideas which have guided the stablishment of the Q-tree. In addition, we define the structures and algorithms for manipulating the Q-tree, we introduce several partitioning strategies based on this structure and, finally, we include the performance results of the different proposals, based on the implementation tasks carried out during the execution of this doctoral thesis.

Índice

Capítulo 1

Introducción

Los sistemas actuales de bases de datos han experimentado un crecimiento exponencial en el volumen de datos a tratar. Este crecimiento ha venido provocado en gran medida a consecuencia de la fuerte competencia existente en distintos sectores de la economía, lo que empuja a las empresas a manejar grandes cantidades de información que faciliten la toma de decisiones en tiempo record. Es por ello que las actuales aplicaciones de bases de datos, de uso cada día más frecuente, tales como GIS, CAD/CAM, CASE y otras, imponen unas exigencias en el campo de la gestión de datos de tal coste que, tan sólo unos años atrás, resultaban imposibles de satisfacer.

Por otra parte, la tecnología relacional, inmersa en la práctica totalidad de los sistemas de bases de datos existentes en el mercado, ofrece a los usuarios y aplicaciones sencillos interfaces que facilitan la expresión de consultas, enmascarando el grado de complejidad de las mismas. Este hecho provoca naturalmente un crecimiento de las expectativas de los usuarios, al tiempo que se somete a la base de datos a un nivel de complejidad mayor.

Las anteriores consideraciones constituyen una importante fuente de requerimientos en los sistemas de bases de datos actuales. La satisfacción de estos requerimientos pasa por la utilización de nuevos métodos y técnicas aplicadas al procesamiento de la información.

La potencia de procesamiento de computadores paralelos económicamente abordables han generado una gran expectativa como solución a los requerimientos mencionados. Debido a ello, los sistemas de bases de datos paralelas han comenzado a desplazar a las tradicionales máquinas de bases de datos construidas sobre grandes computadores ("*mainframes*"). Si bien faltan algunos años para que los sistemas paralelos irrumpan con éxito en el mercado de las bases de datos, la comunidad científica parece haber decantado su interes hacia tales sistemas [19, 59], en detrimento de aquellos otros que, teniendo como soporte un computador con gran potencia de procesamiento y costosos dispositivos de almacenamiento, pretendían resolver los fuertes requerimientos impuestos por el tratamiento y proceso rápido de grandes volúmenes de información.

De acuerdo a la evolución seguida, el futuro de los sistemas de bases de datos parece tender a la utilización de máquinas multicomputador de concepción sencilla, en las cuales se puedan desarrollar modernos mecanismos de paralelización capaces de dar respuesta a los retos planteados. Es en este ámbito en el que se desarrolla la tesis que aquí presentamos.

Con el ánimo de introducir al lector en la problemática y las soluciones originales que aportamos en la presente tesis, este capítulo se organiza como sigue. En el apartado 1 se enuncia la motivación que ha guiado el planteamiento de la tesis. El apartado 2 plantea los objetivos perseguidos durante su desarrollo. El apartado 3 enumera las principales contribuciones de esta tesis. En el apartado 4 se sintetiza la evolución histórica que ha sufrido la investigación llevada a cabo. Por último, el apartado 5 presenta el desarrollo seguido en la escritura de esta memoria.

1 Motivación y ámbito

La adopción generalizada de productos de uso sencillo de la tecnología relacional, ha creado grandes expectativas en la comunidad de usuarios de los sistemas de bases de datos. La aparición de lenguajes de consulta interactivos de alto nivel, tales como SQL, ha reemplazado definitivamente los antiguos hábitos de programación, los cuales requerían un gran esfuerzo de codificación para producir programas que interactuaban con los sistemas de gestión de bases de datos (SGBD) con objeto de generar complejos informes.

Al mismo tiempo, el desarrollo de sistemas multicomputador masivamente paralelos ha evolucionado hasta el punto de concebir máquinas que, a coste razonable, pueden aportar una potencia de procesamiento comparable al de grandes computadores *mainframes*, cuyo coste resulta prohibitivo para la inmensa mayoría de los usuarios. Esta evolución ha favorecido la confluencia de dos campos de investigación que, en el estado actual del desarrollo, se combinan satisfactoriamente para dar respuesta a las necesidades impuestas por un número creciente de usuarios. Hablamos de los sistemas de gestión de bases de datos relacionales (SGBDR) y la tecnología del paralelismo.

La confluencia de ambas tecnologías ha abierto un extenso campo de investigación, cuyas perspectivas de futuro son merecedoras del esfuerzo que actualmente se invierte en el estudio de nuevos SGBDR que adoptan la filosofía del paralelismo como mecanismo básico de funcionamiento.

Fruto de estos esfuerzos son las máquinas actuales de bases de datos paralelas (BDP) más representativas de la literatura, tales como Gamma [17], Bubba [13], Arbre [46], Teradata [15], etc. Sin embargo y a pesar de representar la última generación de sistemas de bases de datos paralelas (SBDP), la mayoría de estos sistemas adoptan esquemas básicos de almacenamiento y acceso a la información que presentan serias limitaciones de rendimiento ante demandas de procesamiento de información, cuyas exigencias desbordan los marcos clásicos de aplicación en los que tales esquemas fueron previstos.

Así por ejemplo, la mayoría de los SBDP mencionados utilizan como método básico de almacenamiento de datos, esquemas de particionamiento horizontal de relaciones sobre el valor de un único atributo. Los fragmentos resultantes de la relación se asignan posteriormente a diferentes nodos de procesamiento con capacidad de almacenamiento secundario. Con tales esquemas de distribución de datos, la ejecución de consultas donde el atributo de particionamiento no aparece como parte del predicado, requiere la participación de todos los nodos que gestionan algún fragmento de la relación particionada, consumiendo por tanto más recursos de los necesarios.

Por otra parte, sistemas como Gamma que prevén el uso de mecanismos de particionamiento basado en varios atributos [27], fundamentan estos mecanismos en la utilización de estructuras multiatributo cuyo grado de rendimiento depende fuertemente de la naturaleza uniforme y equilibrada tanto de los datos propiamente dichos como de los accesos a los mismos.

Las lagunas detectadas en el marco de los mecanismos de almacenamiento y acceso a la información en existentes SBDP, si bien representan el dominio específico de esta tesis, no suponen en modo alguno los únicos problemas abiertos en el extenso ámbito de aplicación que abarcan estos sistemas. Esta circunstancia abre el camino al estudio de nuevas soluciones aplicables a diversos dominios de aplicación estrechamente relacionados con los SBDP.

Fue precisamente en este marco de trabajo donde nuestro grupo de investigación QUATRO [3] recaló, a finales de 1990, con el ánimo de investigar y resolver los problemas inherentes a la construcción de sistemas de bases de datos sobre arquitecturas multicomputador masivamente paralelas. La tesis que se presenta se ubica en el proyecto QUATRO, subvencionado por los proyectos del CYCIT: *Desarrollo basado en sistema Transputer* PROTIC nº IN-88-0149, *Bases de datos gráficas* TIC93-0701-C02-02, y parcialmente desarrollado bajo la *Parallel Computing Action* del proyecto SPRIT PCA-4002.

El objetivo genérico de nuestro grupo se centró en el establecimiento de un sistema relacional de bases de datos soportado por una arquitectura paralela de memoria distribuida, satisfaciendo, en síntesis, los siguientes requisitos:

- Adaptabilidad a cualquier tipo de máquina multicomputador de memoria distribuida.
- Generalidad de utilización como soporte de sistemas de información de propósito no específico.
- Eficiencia en la ejecución de consultas, buscando el mayor throughput del sistema con objeto de explotar al máximo el paralelismo inherente a este tipo de aplicaciones.
- Rapidez de respuesta, explotando eficientemente los recursos disponibles en la ejecución de una consulta.

La aportación de las anteriores propiedades a la máquina QUATRO pasaba por la consecución de los siguientes objetivos preliminares:

- Extensión del modelo relacional para el tratamiento de objetos complejos. En este sentido, se pretendía construir un SGBD relacional con la capacidad añadida de manipular objetos no típicamente relacionales.
- Establecimiento de un modelo de programación específicamente orientado a arquitecturas masivamente paralelas, que aportara modularidad en la construcción del sistema, facilitando además independencia de la máquina física sobre la que se implemente.
- Aportación de un modelo de particionamiento de relaciones que permitiera explotar al máximo el paralelismo, proporcionando eficazmente respuestas a los distintos tipos de consultas emitidas sobre una base de datos de propósito general.

Si bien el primero de los objetivos continúa estando presente en los trabajos a desarrollar por parte del grupo de investigación, los dos últimos han sido prácticamente cubiertos.

Respecto al establecimiento de un modelo de programación, en la tesis desarrollada por nuestro compañero Juan Hernández, "*Modelo de programación basado en actores para sistemas masivamente paralelos*" [33], se presentan las aportaciones realizadas en este ámbito, entre las cuales destaca la definición e implementación de ALBA. ALBA es un nuevo lenguaje de programación concurrente basado en actores que está llamado a ser herramienta fundamental en el desarrollo de nuestra máquina QUATRO.

Por lo que respecta al último de los objetivos originariamente planteados, el establecimiento de un modelo de particionamiento de relaciones para nuestra máquina QUATRO, define el marco inicial de trabajo sobre el que se asienta la presente tesis.

Aparte de la necesidad de un modelo de particionamiento capaz de explotar en su máxima amplitud el paralelismo del modelo arquitectural de QUATRO, la definición de las estrategias de tratamiento de datos soportando el particionamiento de relaciones, requería de la

consecución de una serie de objetivos más concretos, en ocasiones previos y otras veces complementarios, necesarios para implantar con garantías el módulo de acceso a datos de la máquina QUATRO. Estos objetivos se resumen en el siguiente apartado.

2 Objetivos de la tesis

El conjunto de objetivos que han guiado el planteamiento de la tesis que se presenta, se resume básicamente en los siguentes puntos:

1. Establecimiento de una estructura de indexación multiatributo (el árbol Q) capaz de satisfacer las exigencias básicas de acceso a la base de datos QUATRO y ofreciendo características de rendimiento propias de los métodos típicos de acceso a base de datos tales como árboles B. Dicho establecimiento requiere:

   1.a Capacidad de paginación con objeto de mantener un comportamiento eficiente ante la necesidad de su almacenamiento en memoria secundaria.

   1.b Adaptabilidad de uso frente a datos no tradicionalmente involucrados en sistemas relacionales, tales como datos espaciales.

2. Implementación y evaluación de la estructura índice en árbol Q.

3. Planteamiento de un modelo de particionamiento multidimensional basado en la estructura antes mencionada.

4. Comparación del modelo de particionamiento multidimensional frente a modelos clásicos de particionamiento lineal.

5. Proposición de distintas alternativas de particionamiento multidimensional con árboles Q.

6. Análisis y evaluación de las alternativas de particionamiento en operaciones de selección ejecutadas sobre un modelo de máquina paralela de memoria distribuida.

La nominación de los objetivos previos recoge de forma implícita una temporización de la labor realizada durante el desarrollo de la tesis cuya memoria estamos describiendo.

3 Contribuciones

Considerando, por una parte, el creciente interés demostrado por un importante número de investigadores en la explotación de la tecnología del paralelismo en el ámbito de las bases de datos y, por otra parte, la juventud de este vasto campo de aplicación, no resulta difícil encontrar extensas areas vírgenes donde desarrollar ideas noveles que ofrezcan soluciones

originales a los problemas planteados. En este sentido, la tesis que presentamos representa una aportación adicional en una de tales areas, cuya particularidad más significativa es la de conjugar las potencialidades de modernos métodos de acceso con la capacidad del paralelismo para resolver problemas relativos al acceso en una base de datos.

Las contribuciones más destacables resultado de la investigación llevada a cabo en este contexto, pueden clasificarse en dos grandes areas. Por una parte, a lo largo de esta memoria se describen resultados originales relacionados con el mundo de las estructuras de información y métodos de acceso. En el otro extremo de la investigación, se han obtenido modelos originales de particionamiento basados en la utilización de las estructuras de información propuestas. A continuación se detallan los aspectos más reseñables del trabajo de investigación, en cuanto a contribuciones se refiere, en cada una de las areas mencionadas.

3.1 Métodos de acceso

Ubicados en el contexto de las estructuras de información y métodos de acceso, la presente memoria describe una nueva estructura de búsqueda multiatributo, llamada árbol Q. Se trata de un esquema flexible para la recuperación por claves secundarias, el cual cubre con eficiencia todas las clases de búsquedas sobre múltiples atributos.

En comparación a los métodos existentes de acceso multiatributo, el árbol Q incorpora una serie de propiedades únicas que lo caracterizan, las cuales son descritas con detalle a lo largo de esta memoria y aparecen refrendadas por los distintos análisis experimentales llevados a cabo tras la implementación de la misma.

Respecto del trabajo experimental desarrollado en torno al árbol Q, cabe reseñar, aparte del esfuerzo de implementación invertido en la programación del método, la dedicación que ha merecido la generación de baterías de pruebas fiables que permitieran la evaluación del método propuesto. La inexistencia de "*benchmarks*" para la valoración de métodos de acceso, ha supuesto finalmente una necesaria incursión en el campo de la generación de datos sintéticos que, si bien nos ha posibilitado el conocimiento de la problemática implícita, también nos ha supuesto un esfuerzo extra para la preparación de baterías de prueba aptas para valorar las estructuras de búsqueda multiatributo. Los trabajos relativos a la generación de tales baterías están siendo utilizados como herramienta para la evaluación de otras estructuras de indexación multiatributo [60] que son desarrolladas en la actualidad por Antonio Polo en el seno de nuestro grupo de investigación.

3.2 Modelos de particionamiento

Involucrados de lleno en el dominio de la explotación del paralelismo en el campo de las bases de datos, hemos profundizado en el estudio de las distintas alternativas al particionamiento de los datos. Teniendo en cuenta que la práctica totalidad de las máquinas de bases de datos paralelas utilizan un modelo sencillo de particionamiento horizontal basado en la distribución

de valores de un solo atributo, las propuestas que se presentan en esta tesis contribuyen a integrar el concepto de multidimensionalidad, presente en los datos, en la metodología clásica del particionamiento de los mismos.

Las estrategias de particionamiento multidimensional que se describen en esta tesis constituyen una propuesta original cuya principal contribución radica en la capacidad de distribuir equilibradamente los datos de una relación entre un conjunto de nodos, según los valores de múltiples atributos, incluso ante la presencia de un importante sesgo en la distribución de los valores de determinados atributos. A diferencia de los relativamente escasos trabajos sobre particionamiento multidimensional existentes en la literatura, los modelos que aquí presentamos demuestran un aceptable comportamiento ante consultas de distinta procedencia, entre las que se incluyen búsquedas por rango en datos muy sesgados y con una patente correlación entre ellos.

La cuidada selección e implementación de las baterías de pruebas que nos han permitido evaluar la viabilidad de las ideas descritas en esta tesis, constituyen un capítulo importante del trabajo desarrollado. Este trabajo de implementación no sólo supone el punto de apoyo necesario para extraer conclusiones sobre las propuestas que presentamos, sino que además representa una útil herramienta de medida para futuros trabajos emplazados en el ámbito que nos concierne.

4 Evolución histórica de la investigación

Antes de describir la organización de la memoria que describe el trabajo realizado para la presentación de esta tesis, conviene proporcionar una visión retrospectiva de la evolución de las investigaciones realizadas a tal efecto.

La memoria que aquí se describe tiene su punto de partida en los primeros contactos, a finales de 1990, con el profesor Pedro de Miguel. Por estas fechas su grupo realizaba estudios conducentes a la migración de un interfaz de SQL programado para su utilización en una máquina secuencial hacia una arquitectura paralela genérica basada en paso de mensajes.

Después de estos primeros contactos, nuestro grupo de trabajo, constuido por Juan Hernández, Antonio Polo, José Miguel Martínez Candela y el autor de esta tesis, todos pertenecientes al Departamento de Informática de la Universidad de Extremadura, se planteó la posibilidad de aportar nuevas perspectivas de construcción de máquinas de bases de datos paralelas. Este proyecto permitió abrir una vía interesante de investigación lo suficientemente amplia, aunque perfectamente delimitada, para acoger y encauzar sin peligro de dispersión los trabajos que cada uno de sus componentes pudiera desarrollar en un futuro. Es así que nació la máquina de base de datos paralela QUATRO, cuyas lineas genéricas se describen en

[3]. Este anteproyecto inicial y, mas que ello, la ilusión y el reto implícito que representaba, hizo posible el nacimiento de nuestro grupo de investigación, cuyo nombre heredó del proyecto QUATRO.

El informe inicial de QUATRO describía cuatro grandes módulos componentes de la máquina propuesta, de entre ellos, el desarrollo del Gestor de Datos -módulo encargado de llevar a cabo la gestión física de los datos- supuso el punto de partida de la tesis que se presenta.

En la fase inicial de nuestros trabajos comenzaron ya a ponerse de manifiesto algunas de las prestaciones básicas de nuestra máquina a las que debía dar soporte el módulo Gestor de Datos. Entre otras, se consideró prioritario definir con claridad la forma de organizar la información metadato sobre las relaciones fragmentadas. Esto implicaba establecer, por una parte, la manera de particionar las relaciones y decidir, por otra, el tipo de índices con los que se debía trabajar, así como la forma de organizar estos índices en un entorno distribuido.

Tras analizar la bibliografía existente relativa a los SGBDP actuales, se puso de manifiesto una tendencia ciertamente conservadora a la hora de establecer los métodos de acceso utilizados en la práctica totalidad de los mismos. Resultaba de algún modo paradógico que, en el estado avanzado del desarrollo de nuevas estructuras de búsqueda, la totalidad de las propuestas en el campo de los SGBDP utilizaban como mecanismos básicos de acceso a los datos estructuras índice de reconocida eficacia, tales como árboles B. Este patente divorcio en la evolución de areas tan anexas como los SGBDP y las estructuras de información, es lo que nos empujó a considerar la posibilidad de integrar modernos métodos de acceso en sistemas de bases de datos paralelas.

Con tales premisas, nos embarcamos en el estudio de estructuras de información capaces de dar respuesta a consultas frecuentes en los sistemas de bases de datos de popósito general. Esto nos condujo a centrarnos en el estudio de métodos de acceso multiatributo.

El trabajo de navegación por las fuentes bibliográficas en busca de estructuras multiatributo acorde con los requisitos impuestos, culminó en un artículo de Lomet y Salzberg en el que proponían la estructura de árbol hB [44]. Dicho trabajo fue estudiado con exhaustividad, hasta el punto de mantener estrechos contactos con sus autores, durante los cuales se produjeron discusiones relativas a determinados aspectos de la estructura que no quedaban claros en dicho artículo. Finalmente, la complejidad de alguno de los algoritmos de creación del índice, así como la inexistencia de algoritmos para el tratamiento del borrado en el árbol hB, nos llevó a plantear una estructura alternativa de indexación que, si bien recogía algunas de las propiedades del árbol hB y utilizaba en alguna de sus operaciones una metodología similar a la que Lomet y Salzberg proponían, asumía características propias que la diferenciaban con exclusividad. Es así que nació la estructura de árbol Q.

Los primeros resultados relativos a la estructura de árbol Q, revisados personalmente por la Dra. Salzberg, se plasmaron en un informe técnico [5] donde se proponía la estructura y se adelantaban los algoritmos para la creación y mantenimiento del árbol Q.

Con objeto de analizar en detalle el comportamiento de nuestra estructura ante los requerimientos impuestos, llevamos a cabo un arduo trabajo de implementación, sin el cual toda la investigación realizada hasta el momento podría resultar baldía. Algunos meses de dedicación exclusiva a la programación del método de indexación en árbol Q dieron finalmente el fruto deseado. El árbol Q pasó de ser una mera idea teórica sobre la organización del espacio de tuplas, a constituir una seria propuesta de indexación multiatributo cuyos resultados podían ya ser experimentalmente evaluados con datos reales.

El trabajo de implementación del árbol Q nos permitió finalmente componer un artículo [6] para su consideración por la comunidad científica, en el cual se presentaban ya resultados experimentales acerca de la estructura. Con tal fin, dicho artículo se envió a la revista *Information Systems* por recomendación de la Dra. Salzberg. Aunque la evaluación del mismo por parte de uno de los revisores fue positiva, otro revisor detectó un error en uno de los algoritmos de construcción del árbol Q, lo que provocó la desestimación por el cuerpo editoral de la revista para la publicación del mencionado artículo.

El error detectado en tal algoritmo, el cual obviamente no se puso de manifiesto en las pruebas experimentales realizadas con la estructura, fue fácilmente solucionado en la siguiente versión del árbol Q, pero repetir el intento de publicación del referido artículo implicaba una importante demora en nuestros trabajos, ya que en este tiempo estabámos centrados en la confección, implementación y evaluación de los mecanismos de particionamiento de relaciones basados en el árbol Q.

Una vez finalizados los trabajos de implementación y evaluación de la estructura de árbol Q y, tras haber profundizado en el estudio sobre los mecanismos de particionamiento, pronto nos dimos cuenta de las posibilidades que podía ofrecer nuestro método de indexación como soporte para el establecimiento de dichos mecanismos. De este modo surgieron pronto las ideas sobre la explotación del árbol Q en sistemas paralelos dirigidas al particionamiento de los datos. En este caso, la principal dificultad surgía de la imposibilidad de evaluar fehacientemente las estrategias propuestas en una máquina paralela.

Aunque en el seno del grupo QUATRO disponíamos de una pequeña red de transputers, gobernada por el sistema operativo Helios, para efectuar tareas de paralelismo, las enormes dificultades encontradas en la realización de los trabajos experimentales desarrollados por Juan Hernández sobre esta máquina, nos hizo abandonar la primera idea de utilizar la red de transputers para nuestros experimentos.

La carencia de un soporte físico viable donde llevar a cabo nuestros experimentos relativos al particionamiento multidimensional, representó el mayor handicap para la continuación de nuestros trabajos. Esto hizo necesario un trabajo añadido de recopilación de bibliografía con objeto de encontrar la forma de simular un modelo de ejecución de consultas, no excesivamente complejo, que permitiera obtener resultados fiables sobre el comportamiento y la utilidad de los mecanismos de particionamiento que proponíamos.

Finalmente y de acuerdo a los criterios habitualmente seguidos en la confección de los diversos modelos de ejecución de consultas en una base de datos relacional, llegamos al establecimiento de un modelo de ejecución que nos permitió aproximar con cierto grado de realismo, el rendimiento que nuestras propuestas de particionamiento basadas en árbol Q pueden aportar en los SGBDP.

Es conveniente apuntar que, como la mayoría de los trabajos de investigación, la labor que hemos desarrollado en torno a las estructuras de indexación multiatributo y su explotación en SGBDP es susceptible de ampliación por distintas vías. Tareas como la implementación de las operaciones de borrado en el árbol Q; la construcción y gestión distribuida del propio índice en arquitecturas paralelas; su explotación en operaciones más complejas del álgebra relacional (tales como proyección, concatenación, agrupamiento, etc); la evaluación de las propuestas en máquinas paralelas reales, y muchos otros, están aún por concluir. Sin embargo, nuestro grupo de investigación no valora este hecho negativamente, más al contrario, creemos que esta circunstancia abre las puertas para que nuevos miembros que se incorporan al grupo QUATRO, puedan vislumbrar nuevas vías de investigación y formación a partir de las cuales encuentren el resorte necesarioa para desarrollar la fascinante labor de investigación universitaria.

5 Descripción del contenido

En consonancia con los apartados previos, donde desde distintas perspectivas se introduce al lector en la problemática y soluciones que esta memoria plantea, este apartado resume brevemente, a modo de colofón, los contenidos que, capítulo a capítulo, el lector puede encontrar en el interior de la memoria de tesis que se presenta.

El **capítulo 2** ubica con precisión el tema de esta tesis. En las páginas de este capítulo se expone con detenimiento la problemática relativa al campo de investigación en que se enmarca el presente trabajo. En él, hacemos una descripción generalizada de las bases que sustentan los temas relacionados con la tesis y presentamos el estado del arte sobre los mecanismos básicos de particionamiento en bases de datos paralelas y métodos de acceso multiatributo.

La descripción de la estructura de árbol Q se confina al **capítulo 3**. A lo largo del mismo exhibimos no sólo la estructura de indexación con sus correspondientes algoritmos para la creación y utilización del árbol Q sino que, además, justificamos por medio de un conjunto de propiedades rigurosamente formalizadas, las decisiones tanto estructurales como procedimentales tomadas en beneficio de un correcto y eficiente funcionamiento del árbol Q como método de indexación. El capítulo se ilustra con múltiples figuras con objeto de facilitar la lectura y comprensión del mismo.

El **capítulo 4** se centra en la exposición de las propuestas originales para la explotación del árbol Q en SGBDP. Esta explotación muestra dos vertientes claramente viables. Por una parte, la integración del método de indexación en un SGBDP, lo que supone su utilización en paralelo y su gestión distribuida. Por otra, el árbol Q como soporte al particionamiento multidimensional de los datos. En relación a la primera vertiente, se apuntan las lineas sobre las que actualmente trabaja nuestro grupo. Respecto a la segunda, se revelan las alternativas existentes del particionamiento multidimensional basado en la estructura de árbol Q. El capítulo concluye con una descripción del modelo de simulación utilizado en la evaluación de nuestras propuestas.

Los resultados experimentales desarrollados durante la preparación de esta tesis, se reunen en el **capítulo 5**. En él se aportan detalles sobre la implementación de la estructura indexación del árbol Q, la instrumentalización de las distintas baterías de pruebas generadas con el fin de evaluar nuestras propuestas y, por supuesto, los resultados derivados de dicha evaluación, tanto sobre el método de acceso en sí, como acerca del rendimiento de las estrategias de particionamiento multidimensional.

La memoria finaliza con el **capítulo 6**, donde se resumen las conclusiones más destacables y se describen las lineas de trabajo futuro.

Capítulo 2

Modelos de particionamiento y métodos de acceso en bases de datos paralelas

Los sistemas relacionales han dominado el mercado comercial de las bases de datos en la última década, al mismo tiempo que la tecnología relacional ha penetrado en nuevas áreas de aplicación, tales como diseño asistido (CAD/CAM), sistemas de información geográfica (GIS), sistemas de información multimedia (texto, imágenes, voz, vídeo), bases de datos científicas, etc. Comparados con las aplicaciones clásicas de los sistemas tradicionales, básicamente enmarcados en entornos ofimáticos, estas nuevas aplicaciones precisan de tipos de datos más complejos, volúmenes de datos mucho mayores y consultas más complejas. Mientras que los requerimientos básicos para el procesamiento de datos permanecen inamovibles (alto rendimiento, gran fiabilidad y disponibilidad, rapidez de respuesta, etc.), las nuevas aplicaciones suponen un mayor obstáculo para satisfacer en la misma medida estos requerimientos. Resulta entonces necesario aplicar nuevas técnicas para adaptar estos sistemas a las exigencias que imponen.

Los sistemas de bases de datos paralelas (SBDP) han atraído la atención y el esfuerzo de una comunidad importante de investigadores, bajo el convencimiento de que las arquitecturas paralelas constituirán en un futuro próximo una solución realista a los retos planteados en el

ámbito de los sistemas de bases de datos. Es por ello que en la última década una gran cantidad de esfuerzo se ha invertido en la investigación de nuevos sistemas paralelos de bases de datos relacionales (SPBDR).

La presente tesis se enmarca en este ámbito de aplicación, el cual ubicaremos con mayor precisión a lo largo de este capítulo. Con este fin, el presente capítulo se organiza como sigue. En el apartado 1 argumentamos la aparición del paralelismo en el mundo de las bases de datos relacionales. El apartado 2 describe las distintas opciones arquitecturales para construir sistemas paralelos de bases de datos. Las diferentes técnicas de paralelización en aplicaciones de bases de datos son introducidas en el apartado 3. Posteriormente y con el fin de explotar el paralelismo de datos en las aplicaciones que nos conciernen, el apartado 4 detalla los distintos modelos existentes de particionamiento de relaciones y justifica nuestra propuesta de particionamiento multidimensional. Como complemento a todo lo descrito sobre SBDP y con objeto de delimitar la importancia de los métodos de acceso en el trabajo desarrollado en esta tesis, el apartado 5 realiza un repaso por los principales métodos de acceso deteniéndose específicamente en las estructuras índice multiatributo. Finalmente en el apartado 6 aportamos como conclusión una visión sintetizadora de los principales argumentos que nos conducen al planteamiento de la presente tesis.

1 El paralelismo en el ámbito de las Bases de Datos Relacionales

Justificar la aparición del paralelismo en el campo de los sistemas de bases de datos relacionales no es tarea difícil. Las tendencias en el pasado reciente sobre el tratamiento eficiente y rápido de cada vez mayores volúmenes de información, han atraido a lo largo de esta última década una gran cantidad de recursos en investigación para conseguir sistemas de bases de datos de alto rendimiento. Algunas de las causas que exigen un incremento en el rendimiento de los sistemas de bases de datos son las siguientes:

Volumen de datos. En los sistemas actuales, los usuarios precisan mantener grandes volúmenes de datos en linea, de modo que estén disponibles en cualquier momento. Hablar hoy en día de ficheros o tablas con algunos gigabytes no parece sorprendente. La cantidad de datos que se prevé mantener en una simple base de datos en un futuro próximo entra en el rango de terabytes, causando que las consultas lleguen a ser extremadamente intensivas de datos. Además de ello, cada vez resulta mayor la necesidad de integrar modernas aplicaciones en los sistemas de bases datos, tales como diseño asistido por computador (CAD), sistemas de información geográfica (GIS) y aplicaciones multimedia, donde los volumenes de datos son enormemente grandes comparados con las aplicaciones típicas de bases de datos.

Complejidad de las consultas. Como consecuencia de un mercado cada vez más competitivo que demanda una gran cantidad de información sobre la que basar sus decisiones, la complejidad en las consultas emitidas sobre las bases de datos crece día a día. Por otra parte, interfaces más potentes de usuario permiten especificar consultas verdaderamente complejas de forma sencilla, las cuales deben ser respondidas en tiempo real.

Aplicaciones intensivas de datos y lógica. Exactamente igual que ocurre con los interfaces de usuario, nuevos entornos de programación permiten al programador escribir aplicaciones que incluyen un gran número de consultas complejas sobre la base de datos. La combinación de grandes cantidades de datos con capacidades de procesamiento cada vez mayores, favorece sin duda la aparición de aplicaciones que incluyen por una parte gran cantidad de lógica en su programación, tratando por otra parte con un gran volumen de datos

Con tales requisitos resulta comprensible la necesidad de diseñar máquinas de bases de datos con grandes capacidades de procesamiento. Pero ¿por qué los SBDPs han llegado a ser más que una curiosidad en la comunidad investigadora?. Esta pregunta es respondida por DeWitt y Gray en [19] del siguiente modo:

> *"In 1983 relational database systems were just appearing in the marketplace; today they dominate it. Relational queries are ideally suited to parallel execution; they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs."*

Si a esta explicación unimos el hecho de que:

- el costo por MIPS es mucho mayor en computadores de gran potencia que el de pequeñas máquinas basadas en microprocesadores;
- el volumen de datos crece en una proporción similar al crecimiento de las capacidades de memoria principal; y
- a medida que aumenta la potencia de los procesadores, mayor es la distancia que separa la velocidad de procesamiento de la CPU y la capacidad de E/S de un simple dispositivo;

La vía del paralelismo resulta un enfoque muy atractivo para satisfacer los anteriores requisitos en el marco de los sistemas de bases de datos. Las máquinas multicomputador basadas en rápidos microprocesadores, memorias y discos convencionales a precios razonables constituyen un soporte apropiado para ofrecer las capacidades requeridas de procesamiento. La conveniente explotación de diversos mecanismos y estrategias de paralelización en este tipo de máquinas permite ofrecer en conjunto una solución con mayores perspectivas que la que pueden aportar los grandes computadores "*mainframe*". Es ésta la

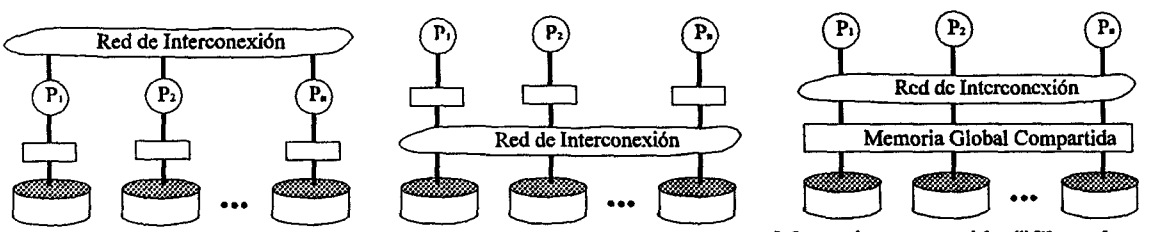
razón de que una ingente cantidad de recursos y esfuerzos en investigación se hayan centrado durante estos últimos años en el campo del paralelismo y su aplicación a los sistemas de bases de datos.

2 Arquitecturas de los sistemas de bases de datos paralelas

Además de dar respuesta a las demandas planteadas de procesamiento, la construcción de un sistema paralelo lleva inseparablemente asociadas las nociones de escalabilidad y aceleración. Un sistema paralelo ideal tiene la propiedad de ser *linealmente escalable*, esto es, doblar la cantidad de componentes físicos tiene el efecto de responder a una tarea dos veces más grande en el mismo tiempo, a la vez que posee una *aceleración lineal*, es decir, doblar la cantidad de componentes físicos supone llevar a cabo la tarea en la mitad de tiempo.

Si bien la escalabilidad y aceleración lineal son propiedades difícilmente alcanzables de modo genérico, el grado de escalabilidad y aceleración representan métricas aceptables para medir el rendimiento y las capacidades de todo sistema paralelo. Con objeto de conseguir sistemas paralelos escalables, se plantean en la actualidad tres alternativas de diseño arquitectural:

- **Sistemas de memoria compartida**. En esta arquitectura, todos los procesadores tienen acceso directo a una memoria principal global común y a todos los discos del sistema.
- **Sistemas de disco compartido**. Esta opción de diseño organiza la memoria principal de forma privada a cada procesador, dejando que todos ellos tengan un acceso compartido directo a cualquiera de los discos del sistema.
- **Sistemas de memoria distribuida**. En este último caso, tanto la memoria como el disco es privado de cada procesador. La única forma posible de comunicación entre procesadores es por paso de mensajes.



Memoria distribuida ("*Shared-Nothing*") Disco compartido ("*Shared-Disk*") Memoria compartida ("*Shared-Everything*")

Figura 2-1. Diferentes alternativas de diseño para sistemas paralelos.

Memoria Distribuida	Memoria Compartida	Disco Compartido
Tandem Non-Stop SQL [71]	XPRS [68]	IBM IMS/VS [69]
Teradata Corp. DBC/1012 [15]	TPF [65]	VAX Rdb/VMS [40]
Bubba [13]	Volcano [28]	DECS [66]
Arbre [46]	DBS3 [11]	
Gamma [17]		

Tabla 2-1. Diseño arquitectural adoptado por sistemas representativos de bases de datos paralelas.

En los tres enfoques anteriores, una red de interconexión de alta velocidad permite conectar los distintos componentes de la arquitectura. La figura 2-1 aporta una visión ilustrativa de los diferentes enfoques de diseño en arquitecturas paralelas.

Aunque existen máquinas de bases de datos representativas para cada uno de los modelos descritos, es prácticamente unánime la consideración actual de la arquitectura de memoria distribuida como la opción de diseño más adaptable a las aplicaciones de bases de datos paralelas. La razón para este consenso no es sino la dificultad de los sistemas de memoria compartida y disco compartido para escalarse convenientemente sobre tales aplicaciones. Por una parte los sistemas de memoria compartida precisan que la red de interconexión tenga un ancho de banda igual a la suma de procesadores y discos, resultando difícil construir redes que puedan escalarse a cientos de nodos. Por otra parte, a medida que crece el paralelismo, la interferencia existente sobre los recursos compartidos limita el rendimiento global del sistema.

El problema de la interferencia presente en arquitecturas de memoria no distribuida puede suavizarse adoptando una configuración de disco compartido. Si la red de interconexión de los discos puede escalarse a miles de discos y procesadores, una arquitectura de disco compartido resulta adecuada para grandes bases de datos de sólo lectura y en general para bases de datos donde no hay compartición de la concurrencia. Sin embargo, para el caso de aplicaciones que leen y escriben en una base de datos compartida, el diseño de disco compartido presenta también problemas de interferencia, debido a la cantidad de mensajes y datos que distintos procesadores deben intercambiar a través de la red actualizando concurrentemente datos compartidos.

Estrechamente vinculado a los problemas de interferencia presentes en arquitecturas no distribuidas, tales enfoques muestran una seria resistencia a la tolerancia a fallos. La compartición de recursos en estos sistemas y por tanto, la dependencia de los procesadores respecto de los recursos comunes, hace complicado divisar mecanismos que permitan al sistema no detenerse ante el fallo de alguno de estos recursos.

Los problemas antes comentados, derivados básicamente de la interferencia sobre los recursos compartidos, en arquitecturas que adoptan un modelo de memoria compartida (ya sea de toda la memoria disponible -memoria compartida-, como de la memoria secundaria -disco compartido-), unido a la capacidad de construir sistemas de memoria distribuida con la adopción de tecnologías más baratas, conducen de forma natural a una extensiva utilización de arquitecturas de memoria distribuida en aplicaciones de bases de datos paralelas, en detrimento de las restantes opciones.

La reflexión anterior, así como la consideración prácticamente generalizada en la comunidad científica sobre las potenciales ventajas de los modelos de memoria distribuida en la construcción de sistemas de gran tamaño [72, 19], avalan en mayor medida la adopción de un modelo de memoria distribuida para el diseño de nuestra máquina de base de datos QUATRO.

El mayor grado de escalabilidad que se consigue con la adopción de arquitecturas de memoria distribuida en aplicaciones de bases de datos paralelas, no está sin embargo libre de costes. A medida que aumenta el número de dispositivos, el equilibrio de carga del sistema se convierte en el punto más crítico, sin cuya participación, los potenciales beneficios de escalabilidad y aceleración casi lineal obtenidos con este tipo de arquitecturas [17, 71] pueden quedar hipotecados. A diferencia de los sistemas de memoria compartida y disco compartido, donde la localización física de los datos tiene una incidencia mínima en el grado de equilibrio de carga del sistema, en los sistemas de memoria distribuida, el grado de equilibrio de carga depende fuertemente de la distribución de los datos sobre los nodos del sistema. Esta estrecha conexión entre distribución de datos y equilibrio de carga, hace necesario el establecimiento de mecanismos de particionamiento y distribución de datos que faciliten la consecución de niveles aceptables de equilibrio de carga en tales sistemas.

A pesar de todo lo mencionado, existen argumentos a favor de una u otra arquitectura según el problema a considerar, de modo que el debate sobre los méritos de los tres enfoques continúa candente. Un análisis de las tres arquitecturas descritas puede encontrarse en [12]. La tabla 1 muestra una selección de sistemas paralelos de bases de datos representativos de cada una de las arquitecturas mencionadas.

3 Técnicas de paralelización en aplicaciones de bases de datos

Una de las principales potencialidades del modelo relacional radica en la característica uniforme que presenta el procesamiento de consultas. En este modelo, cada consulta consiste en un conjunto de operaciones uniformes aplicadas a flujos uniformes de datos. En tal sentido se habla de un procesamiento *orientado a conjuntos*. Cada operador del álgebra relacional

produce una nueva relación, la cual se puede constituir en entrada para el siguiente operador. Esta uniformidad que presentan las operaciones relacionales, hacen del modelo relacional un candidato ideal para introducir técnicas de paralelismo en la ejecución de consultas.

La forma natural de conseguir un alto grado de paralelismo en la ejecución de consultas consiste en particionar los datos de entrada entre múltiples procesadores y memorias. De este modo, un operador relacional puede dividirse en un conjunto de operadores independientes trabajando cada uno sobre una porción distinta de los datos de entrada. Esta forma de paralelismo recibe el nombre de *paralelismo particionado* o *paralelismo de datos*.

Por otra parte, dado que las consultas relacionales pueden ejecutarse como un grafo de flujo de datos, en donde un operador envía su salida a otro, ambos operadores pueden trabajar en paralelo en el momento en que el primer operador comience a ofrecer resultados. A este tipo de paralelismo se le denomina *paralelismo en secuencia ("pipeline")* o *paralelismo de programas*.

Ante la posibilidad de utilizar enfoques diferentes, se plantea de forma espontanea la cuestión sobre cuál de ellos consigue explotar en mayor medida el paralelismo. En este sentido conviene resaltar las limitaciones que impone el paralelismo en secuencia debido a tres factores fundamentales:

- Los grafos de flujo de datos de una consulta relacional raramente permiten una secuencia larga de operaciones (digamos que una cadena de diez operaciones relacionales es bastante inusual).
- Algunos operadores relacionales no emiten su primera salida hasta que han consumido completamente sus entradas (agregados y ordenación son ejemplos de tales operadores).
- Con mucha frecuencia el costo de un operador es mucho mayor que el de otros.

Se desprende de tales consideraciones que el paralelismo de datos ofrece las mayores posibilidades para explotar el paralelismo en operaciones relacionales.

En todo caso, cualquiera que sea la técnica de paralelización empleada en la ejecución de consultas, bien alguna de ellas en solitario o bien una técnica híbrida que incorpore ambos enfoques, el objetivo de paralelización consiste básicamente en reducir el tiempo de respuesta, teniendo como restricción la cantidad de recursos disponibles. Este tiempo de respuesta debería ser reducido hasta un valor *umbral* por debajo del cual la disminución deja de ser significativa. Dicho de otro modo, la idea consiste en maximizar el uso de una serie de recursos limitados disponibles, con el fin de disminuir el tiempo de respuesta de una consulta hasta un valor umbral. Diferentes grados de paralelismo pueden satisfacer este objetivo, de modo que, con el fin de no desaprovechar recursos, es importante encontrar el grado mínimo de paralelismo que satisface el objetivo. Además de esto, el aprovechamiento racional de los recursos disponibles pasa necesariamente por realizar una distribución del trabajo bajo el

criterio indispensable del *equilibrio de carga*. El problema del equilibrio de carga puede establecerse como la capacidad de repartir tan uniformemente como sea posible la carga de trabajo entre las diferentes tareas y recursos físicos disponibles involucrados en la computación.

Frieder y Mason [25] identifican dos categorías de operadores relacionales: operadores *monotuplas* (*"uniscan"*) y operadores *multituplas* (*"multiscan"*). Entre los primeros se encuentran aquellos operadores en los cuales el proceso de cada tupla es independiente del de cualquier otra tupla, tales como los operadores *Select* y agregados. Por contra, se consideran operadores multituplas, aquellos en los que el procesamiento de una tupla involucra datos de otras tuplas. Entre estos últimos se encuentran operadores como *Join* y *Project*.

En los operadores monotuplas, la base del equilibrio de carga se centra en conseguir un particionado inicial de datos tal que en la ejecución de ese operador, cada tarea relativa a un fragmento de la partición, procese un número similar de tuplas. Sin embargo, cuando se trata de un operador multitupla, donde se hace necesario comparar tuplas que pueden pertenecer a fragmentos distintos, una redistribución dinámica de datos es norma habitual para conseguir equilibrio de carga entre los procesadores. Aunque en esta tesis nos centramos básicamente en el operador monotupla *Select*, el cual aparece con gran frecuencia en consultas relacionales, el establecimiento de algoritmos para operadores multituplas satisfaciendo el requisito de equilibrio de carga es motivo de dedicación por parte de nuestro grupo de trabajo. Es importante destacar que el operador *Select* se encuentra prácticamente presente en la inmensa mayoría de los planes de ejecución de consultas relacionales, por consiguiente, un factor de desequilibrio creado por este operador puede perjudicar notablemente la ejecución paralela de la consulta en su totalidad.

4 Modelos de particionamiento de relaciones

Si en el apartado 2 de este capítulo, donde justificábamos la adopción de un modelo de memoria distribuida como base para el desarrollo de nuestros trabajos sobre la máquina QUATRO, aparecía el criterio de equilibrio de carga como punto crítico en el desarrollo de sistemas de bases de datos paralelas, en el apartado 3, la noción de equilibrio de carga vuelve a remarcarse como objetivo necesario en la paralelización de operaciones relacionales. La incidencia de este criterio en el marco de nuestro sistema QUATRO resulta por tanto incuestionable y su consecución, indispensable. A tal fin, se hace necesario establecer mecanismos para fragmentar convenientemente las relaciones, favoreciendo, entre otros, los aspectos de equilibrio en la distribución de datos entre los distintos nodos del sistema. En el presente apartado presentamos las diferentes alternativas al particionamiento de relaciones.

El particionamiento de datos permite a los sistemas de bases de datos paralelas explotar el ancho de banda de E/S de múltiples discos realizando operaciones de lectura y escritura en paralelo. Este mecanismo proporciona un ancho de banda de E/S en muchos casos superior al de sistemas específicos de disco, tales como *disk arrays* sin necesidad de invertir en caros dispositivos con *hardware* especializado.

En sistemas de bases de datos paralelas y distribuidas, existen básicamente dos esquemas de particionamiento: *vertical* y *horizontal.*

El **particionamiento vertical** consiste en construir fragmentos compuestos por varias columnas de la relación y distribuir posteriormente estos fragmentos entre los distintos nodos del sistema. Con objeto de mantener la integridad de la relación, todos los fragmentos deben incluir una columna cuyo valor identifique las tuplas de la relación. Una operación *Join* sobre el atributo identificador permite recomponer la relación original.

En el **particionamiento horizontal**, cada fragmento de la relación contiene un subconjunto disjunto de tuplas completas de la relación. La estrategia del particionamiento horizontal se basa pues en la distribución de las tuplas de la relación original entre el conjunto de discos disponibles.

De entre ambos esquemas, el particionamiento horizontal es el más extendido entre los mecanismos actuales de paralelización de consultas. Esto es principalmente debido a que este último se adapta mejor a la filosofía de paralelismo de datos, cuyo fundamento estriba en la división de las entradas de los operadores relacionales, los cuales actúan siempre sobre una relación para generar una nueva relación. Añadiendo a esto, en primer lugar, que la fragmentación en subconjuntos de tuplas es mucho más homogénea (elementos de distintos subconjuntos son entidades del mismo tipo) y, por tanto, más fácil de conseguir fragmentos de igual tamaño, en segundo lugar, que el particionamiento vertical precisa en muchos casos de una operación extra de *Join* que componga el resultado final de una operación y, finalmente, que este tipo de particionamiento exige la replicación del atributo identificador de cada una de las tuplas en los diferentes fragmentos, resulta comprensible que sea el particionamiento horizontal la opción más interesante en la adopción de un mecanismo de particionamiento. De hecho es el enfoque horizontal el que centra nuestro interés como método para establecer la técnica de particionamiento en el marco de nuestro sistema.

4.1 Estrategias de particionamiento horizontal

En el particionamiento horizontal de las relaciones, se denomina grado de fragmentación o desagrupamiento (*"declustering"*) al número de fragmentos obtenidos tras el particionamiento. Respecto al grado de fragmentación, existen dos posibles enfoques. El primero de ellos, conocido con el nombre de fragmentación total, consiste en partir la relación de tal modo que el grado de fragmentación sea igual al número de discos del sistema. Así por ejemplo Gamma y Teradata adoptan este enfoque para el particionamiento de relaciones.

Contrariamente a la fragmentación total, el grado de fragmentación de una relación puede ser calculado en función de distintos criterios como el tamaño de la relación o la frecuencia de acceso de la misma. Sistemas representativos de este tipo de particionamiento son Bubba y DBS3. Bubba utiliza una técnica original en la cual el grado de fragmentación así como la asignación de los fragmentos sobre los discos, se determina en función del *calor* y la *temperatura* de una relación. El calor de un objeto se define como la frecuencia de acceso a dicho objeto en un periodo de tiempo, y la temperatura viene definida como el calor dividido por el tamaño de dicho objeto. Una descripción detallada de este método puede encontrarse en [16].

Son básicamente tres los tipos de particionamiento más frecuentemente utilizados: particionamiento *cíclico* o *"round-robin"*, particionamiento *desmenuzado* (*"hash"*) y el particionamiento *por rangos*.

El más sencillo de todos es el **particionamiento cíclico**, en el que, las tuplas de la relación se van distribuyendo entre los fragmentos de forma cíclica, esto es, una vez enviada una tupla a cada fragmento, la siguiente tupla se envía de nuevo al primer fragmento y así sucesivamente. Este tipo de particionamiento es idealmente apropiado si las aplicaciones acceden a los datos por medio de accesos secuenciales a la relación. En este sentido, el particionamiento cíclico proporciona un equilibrio de carga inmejorable. El problema con este mecanismo de particionamiento es que con bastante frecuencia las aplicaciones acceden a los datos asociativamente, es decir, buscando tuplas con ciertos valores sobre determinados atributos. Como es obvio el particionamiento cíclico es un tipo de particionamiento ortogonal, en donde la correlación entre la localización de los datos y su uso es mínima. Normalmente, las máquinas de bases de datos que proporcionan particionamiento cíclico, tales como Gamma, lo hacen en conjunción con otros enfoques, dejando la posibilidad de fragmentar una relación según esta estrategia a juicio del usuario.

El **particionamiento desmenuzado** (a veces también denominado particionamiento aleatorio), utiliza una función de desmenuzamiento (*"hashing"*) sobre algún atributo de la relación (habitualmente la clave primaria) para distribuir las tuplas entre los diferentes fragmentos. La función de desmenuzamiento especifica entonces la localización de la tupla sobre un disco particular. Con esta estrategia, el acceso asociativo a la relación buscando tuplas que contengan un determinado valor sobre el atributo de particionamiento, puede ser dirigido a un único disco, evitando así el coste de iniciar tareas sobre múltiples nodos. El problema con este tipo de particionamiento recae precisamente en su aleatorización. En efecto, esta característica de dispersión aleatoria de las tuplas entre los distintos fragmentos, destruye cualquier localidad potencial que pueda existir entre los datos, en concreto, el orden de las tuplas según el atributo de particionamiento es desmantelado tras la aplicación de la función de desmenuzamiento. Debido a la frecuencia con la que aparecen en las consultas determinados patrones de acceso a los datos basados en la adyacencia de los mismos, los datos localmente adyacentes se suelen almacenar físicamente próximos para minimizar accesos a disco (a esta filosofía de almacenamiento se le conoce con el nombre de agrupamiento

("*clustering*")). Es por ello que el particionamiento desmenuzado resulta inapropiado para consultas en las que el predicado incluya algún grado de localidad, tal como orden o proximidad según otros criterios.

Ejemplos de sistemas paralelos que proporcionan este tipo de particionamiento son Gamma, Bubba, Teradata, Arbre, Dbs3 y Prisma/DB.

Por último, el **particionamiento por rango** agrupa las tuplas con similares valores de atributo en el mismo fragmento. Aunque normalmente este tipo de particionamiento se asocia con un orden lineal o lexicográfico sobre alguno de los atributos, cualquier otro algoritmo de agrupamiento puede ser utilizado para conseguir el particionamiento de la relación. Este método de particionamiento sufre el riesgo de un potencial sesgo en los datos ("*data skew*"), ya que un valor concreto del atributo de particionamiento puede aparecer con demasiada frecuencia en las tuplas de la relación, lo que provocaría un desequilibrio en el tamaño de los fragmentos y, por consiguiente, se expone a un potencial desequilibrio de carga en la ejecución de posteriores consultas sobre la relación así distribuida. Este tipo de particionamiento es proporcionado por máquinas tales como Arbre, Bubba, Gamma, Tandem y Dbs3.

Como hemos visto, cada uno de los mecanismos básicos de particionamiento muestran sus potencialidades ante ciertas situaciones, al tiempo que sufren de determinadas debilidades bajo circunstancias diferentes. Sin embargo existen algunos escenarios para los que ninguno de los métodos descritos aportan una solución satisfactoria. Básicamente son los dos siguientes:

- Cuando se ejecuta una consulta en cuyo predicado no aparece el atributo de particionamiento, se hace necesaria la participación de todos los nodos que controlan algún fragmento de la relación. Este hecho puede provocar un desaprovechamiento de recursos, ya que algunos de esos nodos invertirán tiempo y recursos para descubrir probablemente que ninguno de los datos bajo su control satisfacen el predicado de la consulta. En el marco de paralelismo en que nos movemos, estos recursos podrían ser invertidos en tareas más provechosas provenientes de la misma o distintas consultas.
- Considerando el creciente interés por introducir los sistemas de bases de datos en nuevos campos de aplicación, tales como CAD y GIS, en donde los objetos sobre los que se interroga pueden no ajustarse al concepto clásico de tupla en entornos originales de aplicación como sistemas de banca, ofimática y otros, sería deseable buscar modelos de particionamiento que proporcionen una respuesta satisfactoria ante este tipo de nuevos requerimientos. Tal es el caso de objetos espaciales, los cuales se buscan habitualmente por su posición en el espacio, lo cual implica una necesidad de distribuir estos objetos en función de sus propiedades espaciales, mas que según el valor de algún otro atributo específico.

Son precisamente tales lagunas existentes en los mecanismos básicos de particionamiento las que se abordan a lo largo de esta tesis.

Como es lógico, la propuesta de un modelo de particionamiento multidimensional precisa de algún mecanismo capaz de fragmentar los datos de una relación en función de un conjunto de sus atributos. Tales mecanismos son proporcionados por estructuras de datos que organizan el espacio de búsqueda en función de cada una de las dimensiones en las que se enmarca dicho espacio. Son las llamadas estructuras de búsqueda o métodos de acceso *multiatributo* o también *multidimensionales*.

En el siguiente apartado hacemos un repaso de los fundamentos que sustentan el concepto de estructuras de búsqueda multidimensionales, así como de las principales estructuras índice multidimensionales existentes.

Conviene resaltar que a pesar de existir una amplia variedad de mecanismos de indexación multiatributo basados en esquemas de desmenuzamiento [37, 64, 54, 56, 70], tales estructuras no son objeto de tratamiento en el siguiente apartado debido principalmente a la inoperancia de tales estructuras ante consultas de tipo rango. Como fue mencionado con anterioridad las estructuras de indexación basadas en el desmenuzamiento destruyen cualquier localidad potencial que exista entre los datos que se desean indexar. Esta característica propia de los esquemas de desmenuzamiento aleja nuestro interés en las mismas.

5 Estructuras de datos multidimensionales

Todos los sistemas de bases de datos proporcionan mecanismos de indexación que favorecen el acceso selectivo a una gran colección de registros de forma asociativa. La gran mayoría de tales sistemas proveen al usuario con la capacidad de construir índices sobre cualquiera de los atributos de la relación. Con frecuencia un índice agrupado según los valores de clave primaria constituye una imposición por parte del sistema y, adicionalmente, el usuario puede construir índices no agrupados sobre otros atributos de la relación.

Normalmente un usuario puede decidir crear un índice sobre varios atributos de la relación, pero habitualmente la organización de los datos que proporciona este índice se basa en un orden lexicográfico según la posición del atributo en la lista de atributos que se desea insertar. Por ejemplo, un usuario puede decidir construir un índice sobre los atributos *Primer_Apellido*, *Segundo_Apellido* y *Nombre* en una supuesta relación con información sobre individuos. En este caso, el método de acceso proporciona una visión ordenada del fichero en función del primer apellido y, dentro de aquellos individuos con el mismo primer apellido, se mantiene el orden impuesto por el segundo apellido, de modo que aquellos individuos con iguales apellidos se perciben ordenados según su nombre. Como es obvio, un índice tal no aporta facilidad alguna para realizar búsquedas por el segundo apellido ni por el nombre. Tales índices no se consideran multiatributos.

Usualmente, la vía que proporcionan los actuales sistemas de bases de datos para permitir búsquedas por varios atributos es mediante la construcción de listas invertidas, en las que un índice para cada clave (atributo) sobre la que se desea buscar soporta la percepción del fichero ordenado en función de dicha clave. En realidad las listas invertidas son extensiones de estructuras de búsqueda originalmente diseñadas para acceso por clave única. Las listas invertidas, aparte de precisar gran cantidad de espacio para su almacenamiento no resuelven convenientemente el problema de la adaptación a ficheros altamente dinámicos. Además, estas estructuras no son válidas para organizar datos de tipo espacial, ya que cada índice secundario en la lista invertida aporta una ordenación diferente de los datos en función del atributo (dimensión) que indexa, no favoreciendo una visión unificada del espacio de búsqueda.

Nuestro interés se centra en estudiar estructuras índice específicamente diseñadas para soportar el concepto de orden global o localidad en el espacio multidimensional de los diferentes elementos que se engloban en dicho espacio de búsqueda. Nos adentramos pues en el campo de las estructuras índice multiatributo.

5.1 Fundamentos de la indexación multiatributo

La idea que subyace en la indexación multidimensional se describe fácilmente considerando cada tupla t de una relación $R(A_1, A_2, ..., A_d)$, como un punto en el espacio d-dimensional $t = (k_1, k_2, ..., k_d)$, donde k_i es un valor del dominio de A_i. Una estructura de búsqueda capaz de particionar organizadamente el espacio multidimensional es, por tanto, susceptible de ser utilizada como método de acceso multiatributo, ofreciendo facilidades de acceso a las tuplas de la relación por uno o varios de sus atributos.

Una gran variedad de métodos de acceso multiatributo han sido propuestos a lo largo de estas dos últimas décadas. Existen básicamente dos vías distintas para aportar soluciones en el marco de la indexación multiatributo:

- Representación del orden multidimensional en el espacio unidimensional.
- Particionamiento del espacio multidimensional y su representación por un índice multiatributo.

La primera de ellas consiste en establecer una correspondencia del espacio multidimensional sobre el espacio unidimensional, definiendo un "camino" lineal a través de un espacio multidimensional. El más claro exponente de esta línea se define en [53]. Orenstein y Merret definen tal camino a través de un "Z-orden", entrelazando los bits que representan los valores de distintos atributos en una única cadena de bits. En este caso, un método de indexación unidimensional, tal como el árbol B^+, puede ser utilizado para conducir la búsqueda en el espacio multidimensional. La dificultad de este enfoque es que la especificación natural de los rangos de búsqueda multiatributo debe ser traducida a una colección de rangos en el espacio unidimensional imagen de la correspondencia.

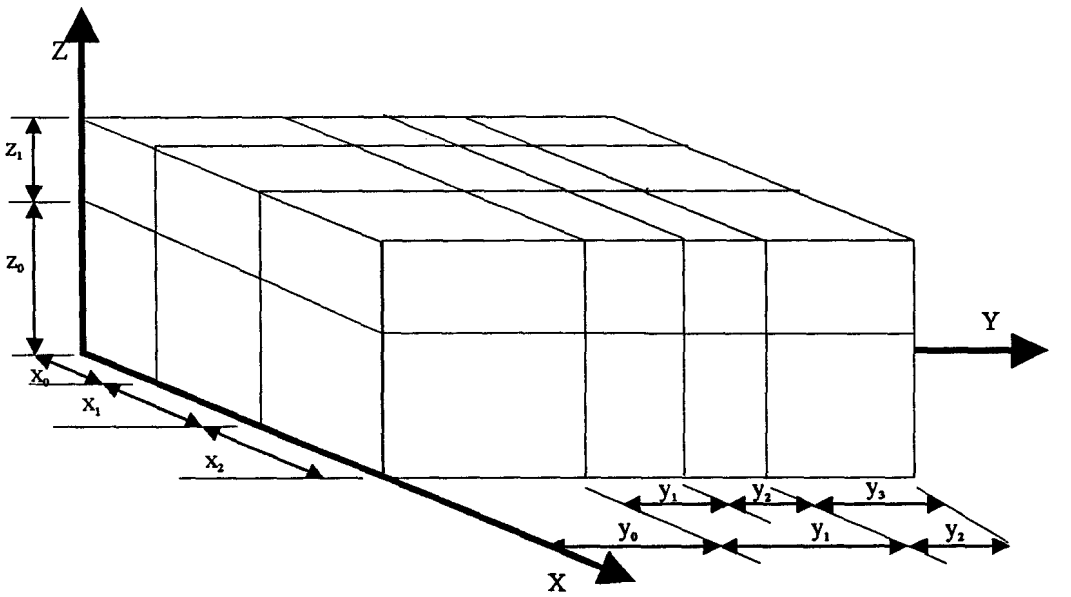


Figura 2-2. Organización del espacio de búsqueda según se percibe por un fichero rejilla.

La segunda vía, para la cual existen muchas propuestas en forma de métodos de acceso multiatributo, se fundamenta en dividir progresivamente el espacio de búsqueda, obteniendo finalmente un particionamiento organizado del mismo, en el cual cada partición agrupa aproximadamente un mismo número de puntos cercanos en el espacio. Este particionamiento se realiza habitualmente eligiendo una dimensión (atributo) conveniente y un valor frontera que deje los puntos del espacio a ambos lados del hiperplano que definen dicha dimensión y dicho valor frontera. Tal división produce una asignación de puntos a los subespacios resultantes relativamente uniforme. En ocasiones, es posible que muchos puntos se encuentren sobre el hiperplano de división, lo cual aumenta la complejidad del problema. Sin embargo la dificultad real de estos métodos consiste en organizar, siguiendo la misma filosofía de particionamiento, los elementos del índice.

A medida que el espacio de búsqueda se va dividiendo, el índice va registrando las distintas divisiones. Mientras que los datos del espacio de búsqueda son puntos, los términos del índice representan entonces subespacios con fronteras más complejas. Estas fronteras complejas complican en mayor medida el proceso de encontrar una cota sencilla que delimite claramente los términos índice en regiones disjuntas, de forma que pueda continuarse aplicando progresivamente la misma técnica de indexación a los elementos del índice.

Enmarcados en esta segunda vía, los esfuerzos por proponer distintas alternativas de diseño de estructuras índice han seguido dos líneas distintas:

- Organizaciones basadas en ficheros rejilla.
- Estructuras árboles generalizadas al espacio multidimensional.

Los siguientes subapartados pretenden ofrecer una visión genérica de las distintas soluciones y sus implementaciones más representativas con objeto de delimitar con mayor claridad el marco de esta tesis en lo referente a las estructuras índice multiatributo.

5.1.1 Ficheros rejilla

Las organizaciones basadas en ficheros rejilla tienen su punto de partida en la estructura de fichero rejilla ("*grid file*") propuesta por Nievergelt, Hinterberger y Sevcik en [52]. La idea que subyace en su propuesta consiste en considerar el espacio particionado en forma de red o rejilla, de modo que un valor para una dimensión concreta divide todo el espacio. El espacio de búsqueda puede percibirse entonces como un conjunto de cajas multidimensionales, tal y como muestra la figura 2-2. Cada dimensión se encuentra dividida en un conjunto de intervalos de tamaño no necesariamente regular. En respuesta a nuevas inserciones, la partición se refina alterando sólo una de sus coordenadas a un tiempo. La figura 2-2 muestra este refinamiento en la dimensión X. Cada cubo multidimensional representa un bloque de disco donde se almacenan las tuplas correspondientes a los puntos dentro del cubo.

En el caso del fichero rejilla, el índice a este espacio particionado consiste en un directorio que representa la partición del espacio por medio de un array multidimensional conteniendo apuntadores a los bloques y un conjunto de escalas (tantas como dimensiones tenga el espacio de búsqueda) conteniendo las cotas de los intervalos en que se divide cada dimensión. La figura 2-3 ilustra los distintos componentes del fichero rejilla en el espacio bidimensional. Cuando el tamaño del fichero es excesivamente grande como para mantener el directorio en memoria principal, una versión a escala reducida del directorio en memoria principal puede ser construida con objeto de indexar dicho directorio en disco.

El principal problema que presentan los esquemas de indexación basados en ficheros rejilla es la intensiva replicación de apuntadores a bloques que puede llegar a incluir el directorio. Esta replicación es consecuencia de la forma en que se ejecuta la división del espacio cuando

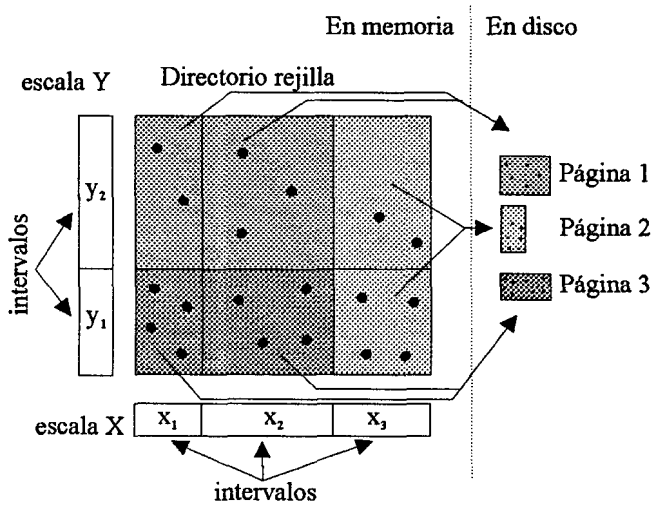


Figura 2-3. Diferentes componentes de la estructura de un fichero rejilla.

un bloque de disco se sobrecarga. Ante una situación de sobrecarga, un valor para una dimensión específica corta todo el espacio a lo largo de dicha dimensión. Esto provoca una división en cadena de los cubos en el directorio, de modo que todos los elementos del directorio que se vean "atravesados" por el corte efectuado deben ser duplicados, salvo aquél que representaba al bloque sobrecargado. El problema lógicamente se agrava con el tamaño del fichero.

La existencia de tanta replicación de apuntadores en el directorio impide aplicar una técnica progresiva de división del propio directorio que sea equilibrada, es decir, tal que tras la división del directorio, los términos índices resultantes indexen un número similar de bloques de datos.

A pesar de estas desventajas, el potencial de esta estructura de indexación ha merecido el interés de muchos investigadores, lo que se ha traducido en la propuesta de algunos derivados del fichero rejilla [24, 36], así como de esquemas de particionamiento de relaciones que toman como base esta estructura [27, 63].

5.1.2 Indices multiatributo basados en árbol

Basada en una filosofía similar pero siguiendo un método de implementación distinto, un amplio conjunto de estructuras índice basadas en alguna forma de representación arbórea aportan solución al problema de la indexación multiatributo.

Este conjunto de estructuras multiatributo, utilizan alguna forma de árbol para representar una división jerárquica del espacio de búsqueda. Así por ejemplo, los árboles k-d de Bentley [8] utilizan un atributo y un valor adecuado sobre ese atributo para realizar la partición de una región. Posteriormente, cada región resultante se divide separadamente utilizando un nuevo atributo de la relación. La figura 2-4 aporta una visión de la organización del espacio bidimensional siguiendo esta filosofía. Cada una de las regiones producto de la división debe contener un número similar de puntos. El principal inconveniente que plantean los árboles k-d es su potencial desequilibrio y falta de paginación, lo cual impide un almacenamiento equilibrado y unos tiempos de acceso razonables a la estructura cuando no cabe enteramente

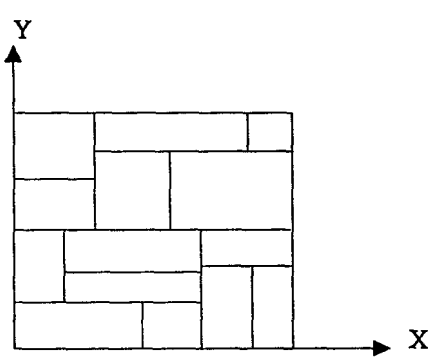


Figura 2-4. Organización del espacio bidimensional según el árbol k-d.

en memoria principal.

Por su parte, los árboles cuaternarios ("*quad trees*") [23], localizan un punto a partir del cual realizan una división en 2^d regiones. Igual que antes cada vez que una región se sobrecarga, un nuevo punto interior a dicha región funciona como separador de un nuevo conjunto de 2^d regiones, siendo d el número de dimensiones. El nodo interno del árbol contiene la información del punto y las 2^d regiones son sus hijos. La figura 2-5 ilustra esta filosofía de división del espacio bidimensional. Su principal desventaja radica en su impracticabilidad cuando el número de dimensiones es alto, debido a que los nodos del árbol llegan a consumir mucho espacio y cada nodo tendrá un número alto de hijos nulos, es decir muchas de las regiones resultantes no contendrán tupla alguna. Además sufre de la misma debilidad del árbol k-d en cuanto a su paginación.

Existen otros trabajos sobre indexación multiatributo cuyo fin es la generalización a múltiples dimensiones del ubicuo árbol B (o cualquiera de sus derivados). Así por ejemplo los árboles B multidimensionales de Ouksel y Scheuerman descritos en [55] consisten en un árbol de tantos niveles como claves se desean indexar. Cada nivel consiste a su vez en un conjunto de uno (la raíz) o más árboles B de distinto orden y tamaño. Este tipo de estructuras incluyen una gran cantidad de apuntadores, tanto a hijos como a nodos hermanos y conllevan gran complejidad en su tratamiento. Otras estructuras representativas de esta filosofía de generalización del árbol B pueden encontrarse en las referencias [31, 39].

Robinson es uno de los primeros en proponer una estructura multiatributo paginada que intenta reunir la eficiencia de búsqueda multidimensional de los árboles k-d equilibrados [10] junto con el buen comportamiento en operaciones de E/S de los árboles B. Se trata del árbol K-D-B [62].

El árbol K-D-B distingue dos tipos de nodos: nodos de datos (hojas) y nodos índice (internos). Los nodos de datos contienen colecciones de registros y representan regiones

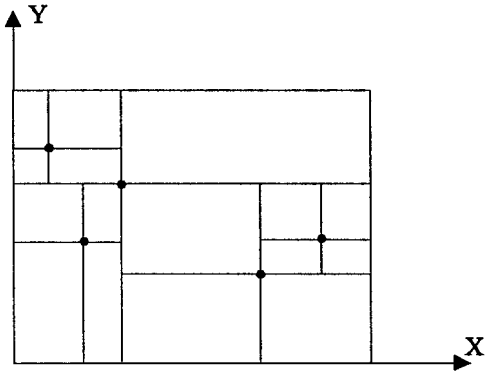


Figura 2-5. División del espacio en el árbol cuaternario.

disjuntas del espacio de búsqueda. Los nodos índice contienen términos índice que son pares

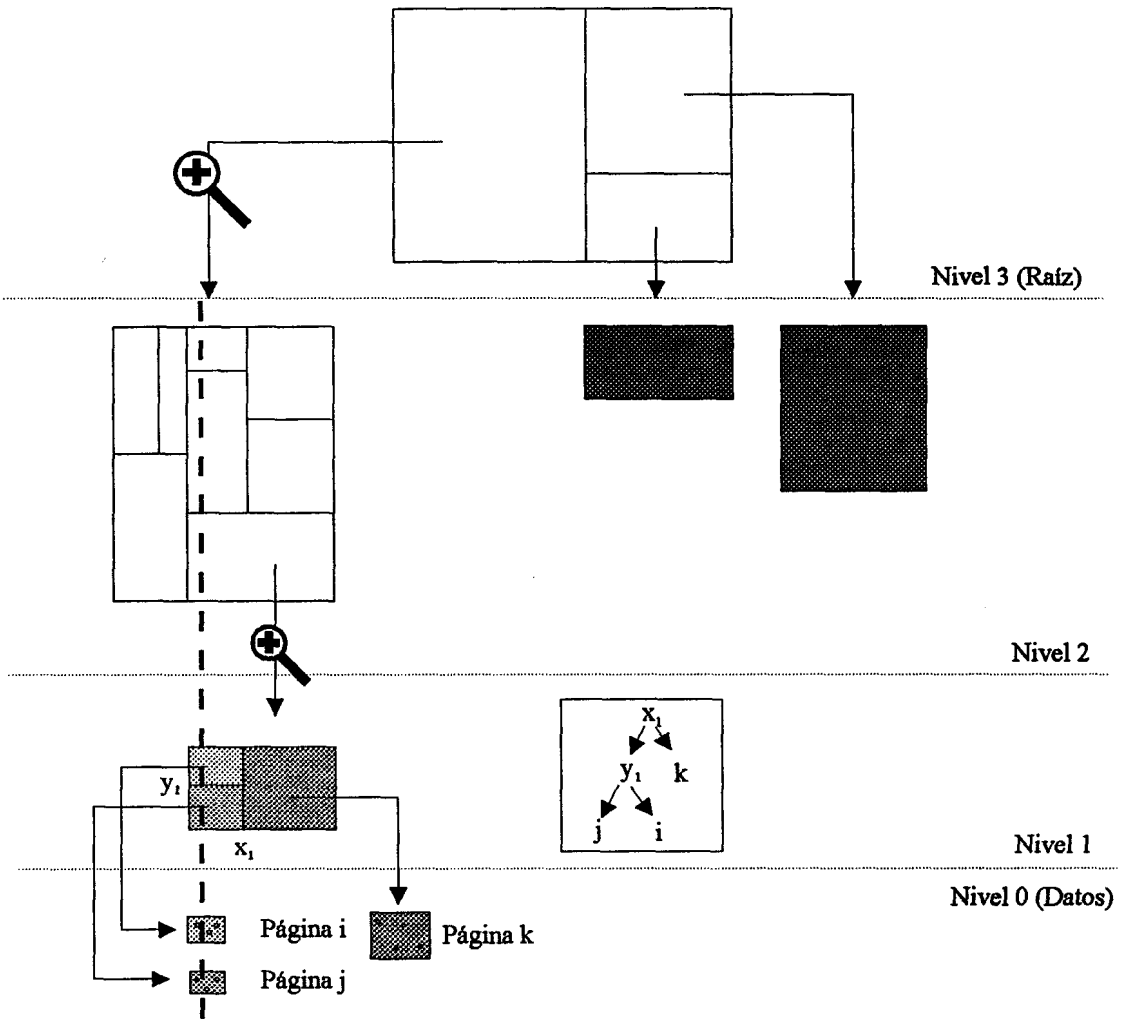


Figura 2-6. Organización y división del espacio según la estructura de árbol K-D-B. La división de un nodo índice provoca divisiones en cascada (línea discontinua).

de la forma (región, ID-página). En realidad un nodo índice registra la forma en que se encuentra dividida la región que referencia. Esta información de división se almacena en forma de árbol k-d.

En la figura 2-6 se ilustra la noción de árbol K-D-B en un espacio bidimensional. Como puede deducirse de la figura, cada nivel se refina progresivamente hasta llegar a las páginas de datos. La organización interna de los nodos del índice se representa por medio de un árbol k-d.

El problema fundamental de esta estructura índice es que la división de un nodo índice puede provocar una división en cascada de nodos índices y páginas de datos. Esto se debe al hecho de que la división de un nodo índice se efectúa por un hiperplano que corta a todo el espacio y no sólo a la región que el nodo índice a dividir representa. La línea discontinua vertical en la figura, representa la división de un nodo índice a nivel dos y la consiguiente división en cascada de los nodos índices y datos descendientes que son cortados por la línea.

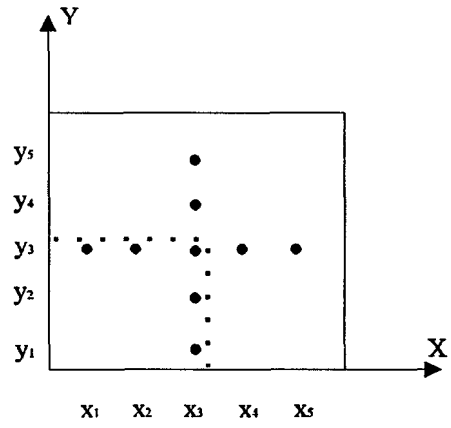


Figura 2-7. Distribución de puntos en el espacio. No se encuentra un hiperplano que divida uniformemente el conjunto de puntos en dos partes.

Tomando como base la organización que proponen los árboles K-D-B, Lomet y Salzberg divisan una nueva estructura multiatributo a la que dan el nombre de árbol hB [44]. El índice en árbol hB resuelve el problema de la división en cascada y propone una división del espacio de búsqueda por más de un atributo en el caso en que esto sea necesario. Para aclarar esta circunstancia, sus autores argumentan que, bajo ciertas distribuciones de puntos en el espacio, no es posible encontrar un hiperplano que divida al espacio en dos regiones con aproximadamente el mismo número de puntos. La figura 2-7 ilustra un escenario así. En este caso ningún valor de X o Y establece una división sobre el espacio en dos regiones dejando en cada una de ellas un número similar de puntos. Sin embargo usando, en el caso presente, los valores de ambos atributos, podríamos establecer una división uniforme de puntos. El rectángulo de líneas discontinuas en la figura marca dicha división. Usando la terminología original, esta división puede percibirse como si al espacio inicial considerado como un "ladrillo" (*"brick"*) le hiciéramos un agujero (*"hole"*), resultando finalmente un espacio formado por ladrillos posiblemente agujereados (*"holey bricks"*). Esta visión de cómo organizar el espacio es la que le presta el nombre a los árboles hB.

Lomet y Salzberg demuestran en [44] que, en un espacio d-dimensional, con la intervención de a lo sumo d atributos siempre es posible encontrar una forma de dividir una región en dos partes con un ratio de división en el peor caso de 2:1, esto es, de modo que 1/3 de los puntos caigan a un lado y los restantes 2/3 al otro.

Desde un punto de vista estructural, el árbol hB se compone de dos tipos de nodos: nodos índice y nodos de datos. Los nodos de datos contienen las tuplas (puntos) y están internamente asistidos por un árbol k-d local que organiza la página en listas de registros según los valores de varios atributos. Por su parte, los nodos índice se organizan en forma de árboles k-d que representan la forma en que se divide la región que cada nodo índice gobierna. De la misma forma en que se divide una página de datos, la sobrecarga de un nodo índice conlleva una división de la región que dicho nodo indexa. Esta división se efectúa extrayendo un subárbol que representa un ladrillo dentro de un ladrillo posiblemente agujereado (representado por el

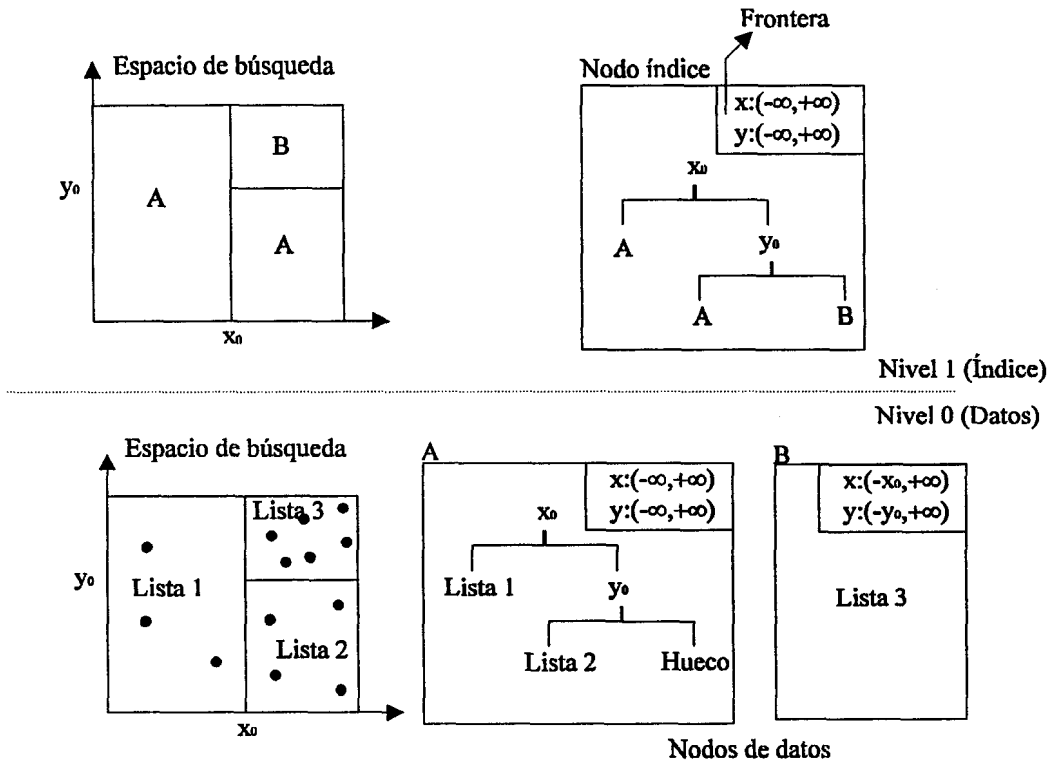


Figura 2-8. Organización del espacio según la estructura de árbol hB. Mientras que el nodo B es un "ladrillo", el nodo A es un "ladrillo agujereado".

árbol k-d local al nodo índice). Como resultado de la división se obtienen dos regiones, una en forma de ladrillo, la región extraída, y otra en forma de ladrillo agujereado (la antigua región sobrecargada una vez extraído el subárbol). La figura 2-8 muestra una representación del espacio cuya organización está asistida por un árbol hB.

Esta reciente estructura índice multiatributo que incorpora ideas inteligentes sobre el modo de conseguir una organización incremental del espacio a medida que nuevos datos van siendo insertados, comprendía no obstante complicados algoritmos de división de nodos índice y traslado de términos índice hacia los nodos ascendentes, los cuales (debido posiblemente a su complejidad) se demostraron posteriormente erróneos en [22].

En un proceso posterior de corrección de los problemas de la estructura de árbol hB e incorporación de propiedades de concurrencia y recuperación a dicha estructura, Evangelidis propone en [22] un nuevo método de indexación llamado árbol hB^{Π}.

Conviene reseñar aquí el interés que nuestro grupo de trabajo mostró por aquella nueva estructura de árbol hB como posible candidata para nuestro modelo de base de datos QUATRO. Este interés nos hizo profundizar en aquellas ideas y mantener una vía fluida de comunicación con sus autores, al objeto de poder resolver las dificultades que planteaba su utilización sin modificaciones en nuestra máquina de base de datos como método básico de acceso. Esta colaboración condujo por una parte al establecimiento y fijación de ciertos

problemas estructurales en el árbol hB que fueron descubiertos por nuestro grupo y posteriormente resueltos por Evangelidis, tal y como él mismo aduce en [22] y, por otra, al planteamiento por nuestra parte de la estructura de árbol Q, la cual tiene como punto de partida el trabajo de Lomet y Salzberg en [44].

Por último es preciso destacar los esfuerzos realizados por otro componente de nuestro grupo de trabajo con objeto de generalizar la estructura de árbol Q que ahora presentamos. Esta generalización ha llevado al establecimiento de una nueva estructura que, bajo el nombre de árbol mQ (árbol Q multidimensional), aporta buenos resultados en su explotación ante el conjunto básico de operaciones relacionales.

5.2 Estructuras multiatributo para datos espaciales

En el ámbito de datos espaciales, los cuales en ocasiones los datos no son simples puntos en el espacio sino por ejemplo objetos geométricos (frecuentemente áreas o volúmenes), la idea básica más frecuente para organizar el espacio consiste en representar tal objeto por medio de su "*caja de acotación*". La caja de acotación de un objeto es un rectángulo multidimensional, compuesto por d intervalos cerrados $[a,b]$ (siendo d el número de dimensiones) que describen la extensión del objeto a lo largo de la dimensión correspondiente. De este modo cada objeto espacial puede ser considerado como una d-tupla cuyos valores de atributo son rangos para una dimensión.

Lomet distingue en [45] dos estrategias básicas para organizar datos espaciales y aportar métodos de acceso a datos de este tipo: por indexación del espacio nativo y por indexación del espacio transformado.

La indexación del espacio nativo preserva la distancia métrica del espacio original, es decir, los objetos próximos en el espacio nativo también se encuentran próximos en el espacio indexado. Debido a que los objetos espaciales se solapan, puede no ser posible encontrar una división limpia que delimite el espacio global en subespacios que no se solapen. Esta característica provoca la necesidad de acudir a investigar en varios subespacios en busca de un punto que se encuentre dentro de una región solapada. Estructuras típicas que adoptan esta estrategia de indexación son los árboles R [32] y algunos de sus derivados, como el R^+ [67].

La segunda estrategia propone considerar cada coordenada de la caja de acotación de un objeto como un punto en el espacio bidimensional. En este caso, la cota inferior y superior de cada coordenada de la caja se usan como atributos de indexación. De este modo es posible usar un método de indexación multiatributo, en donde cada tupla se percibe como un punto en el espacio $2d$-dimensional, siendo d el número de dimensiones del espacio nativo de búsqueda. El espacio nativo se transforma convirtiéndose en un espacio con el doble de dimensiones. Este mecanismo básico de indexación permite además que otros atributos no espaciales puedan ser indexados del mismo modo que el resto.

La correspondencia que se establece entre las fronteras de la caja de acotación y los puntos en el espacio $2d$-dimensional tiene algunas propiedades interesantes a destacar. Si dos puntos en el espacio $2d$-dimensional poseen valores similares en todas las coordenadas, los correspondientes objetos d-dimensionales son similares en tamaño. Si son pequeños se encuentran próximos en el espacio y si son grandes se superponen. Con tales propiedades, un método de indexación multiatributo de espacio transformado que agrupe puntos cercanos, agrupará en páginas grandes objetos espaciales que se solapen. Igualmente, un método así, facilitará la agrupación de pequeños objetos que se encuentren próximos en el espacio nativo.

Para ilustrar las propiedades de esta correspondencia, consideremos objetos espaciales unidimensionales consistentes en segmentos de linea con un valor inicial y un valor final. Tales objetos pueden mapearse en un espacio bidimensional, donde la coordenada x se corresponde con el valor inicial del segmento, y la coordenada y con el valor final del mismo. Como es obvio, todos los puntos en el espacio imagen bidimensional se encontrarán por encima de la diagonal $x = y$. Aquellos objetos que representen segmentos cortos de linea se corresponderán con puntos cercanos a la diagonal, mientras que los segmentos largos de lina se mapearán en puntos alejados de dicha diagonal. Esta característica permite que las consultas usuales en el ámbito espacial, tales como inclusión, intersección o disjunción, puedan representarse en el espacio transformado mediante rectángulos $2d$-dimensionales. Por ejemplo, en la figura 2-9, todos los segmentos de linea que contienen al segmento [3,5], representado por el punto (3,5), se encuentran en el area A, aquellos incluidos en este segmento se localizan en el area B, los segmentos con que interseca se encuentran en el area C, y los segmentos disjuntos a éste se encuentran en el area D.

Esta capacidad de agrupamiento es muy útil en consultas espaciales, ya que los grandes objetos satisfarán con frecuencia un alto número de búsquedas, por lo que si se mantienen agrupados, se precisarán menos accesos a disco, incrementando la localidad de referencia. De igual modo, la búsqueda de pequeños objetos requerirá un menor número de páginas accedidas para responder a una consulta.

Teniendo en cuenta nuestro interés por aportar una estructura de datos multidimensional fundamentalmente aplicable en entornos típicamente relacionales, pero extensible a objetos espaciales, es este segundo enfoque el que nos interesa como vía para tratar tales objetos mediante estructuras índices multiatributo.

Aunque la aplicación a datos espaciales de la estructura que proponemos en el desarrollo de la presente tesis queda fuera del marco de la misma, constituyendo una línea de trabajo para desarrollo futuro, la conclusión más importante que extraemos de este apartado es que las estructuras multiatributo constituyen una herramienta de indexación en la que concurren excepcionales potencialidades para su aplicación en el campo de tratamiento de datos espaciales. Un método de acceso multiatributo capaz de ofrecer rendimientos similares al de estructuras índice monoatributos, tales como árboles B, en aplicaciones típicamente relacionales, se convierte en un candidato excelente para su aplicación en modernos sistemas

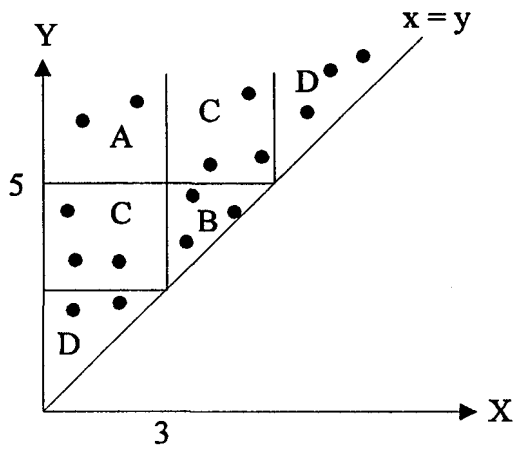


Figura 2-9. Correspondencia de segmentos de linea con puntos en el espacio bidimensional.

relacionales que en un futuro cercano deben satisfacer requisitos propios de aplicaciones intensivas de datos como GIS, CAD/CAM y otras en las que el tratamiento de datos espaciales cobra especial relevancia.

6 Conclusiones

En el presente capítulo hemos delimitado el ámbito de aplicación de la tesis que, capítulo a capítulo, iremos desgranando a lo largo de esta memoria.

Las ideas que proponen la vía del paralelismo como solución a los mayúsculos requerimientos que día a día imponen las nuevas aplicaciones, recalan con prontitud en el campo de las bases de datos. De modo más específico, los sistemas relacionales resultan ser muy suceptibles para su adaptación al paralelismo, por lo que de inmediato se abre una extensa linea de investigación aplicada, que comienza a aportar sus primeros frutos con el establecimiento de los primeros propotipos de máquinas de bases de datos paralelas, mediada la década de los 80 (en [33] puede encontrarse un extenso trabajo de recopilación donde se muestra la evolución de los sistemas de gestión de bases de datos en general, en el que se destacan, con gran lujo de detalles, las características de estos sistemas y, en particular, de las máquinas paralelas). De entre las arquitecturas subyacentes capaces de soportar un modelo de máquina paralela, es el modelo de memoria distribuida, por sus potencialidades en lo que a escalabilidad se refiere, el que atrae las mayores atenciones como arquitectura básica para los futuros sistemas masivamente paralelos.

A pesar de las indiscutibles ventajas que el modelo de memoria distribuida aporta frente a las alternativas de memoria compartida y disco compartido, la consecución de un alto grado de equilibrio en la carga de un sistema con soporte arquitectural de memoria distribuida, se presenta como un objetivo primordial y básico para conseguir los beneficios esperados de escalabilidad y aceleración lineal del sistema.

Si consideramos por otra parte que el paralelismo de datos se presenta como la forma más natural de explotar el paralelismo en la ejecución de los operadores relacionales, el problema del equilibrio de carga en un modelo de memoria distribuida -donde los nodos pueden trabajar en paralelo, ejecutando una misma operación sobre diferentes partes de los datos de entrada-, se encuentra estrechamente relacionado con la capacidad de particionar convenientemente las relaciones sobre las que se ejecuta la operación.

Con tal objeto, en el marco de este capítulo se repasan los principales métodos de particionamiento de relaciones que, una vez valorados, dejan al descubierto una serie de deficiencias cuando son analizados bajo el prisma de determinados requerimientos que centran nuestro interés. Toda vez que entre los requerimientos mencionados se encuentra la utilización de varios atributos como claves de búsqueda en el acceso a los datos, así como la posibilidad de trabajar con datos espaciales, nos introducimos en el estudio de estructuras de datos multidimensionales.

De acuerdo al planteamiento previo, en el seno de este capítulo hacemos un repaso sobre las principales estructuras soporte de la indexación multiatributo. La estructura de ficheros rejilla, así como las más modernas organizaciones de índice basadas en árboles son examinadas y valoradas desde la perspectiva de su utilidad como soporte al particionamiento multidimensional, a la vez que mecanismo básico de indexación para nuestra máquina QUATRO.

Las principales debilidades de los métodos examinados, puestas de manifiesto a lo largo del estudio realizado, nos conducen finalmente al establecimiento de un nuevo método de indexación multiatributo: el árbol Q.

En el capítulo siguiente describimos con detalle este método original de indexación. Las estructuras necesarias para su instrumentalización, así como las propiedades convenientemente formalizadas y los algoritmos para la construcción y el crecimiento de nuestro árbol Q, son objeto de desarrollo en el siguente capítulo de la presente memoria.

Capítulo 3

El árbol Q

Tal y como argumentamos en el capítulo anterior, la propuesta de un modelo de particionamiento multidimensional de relaciones capaz de explotar el paralelismo en modernas aplicaciones de bases de datos relacionales, precisa del establecimiento de una estructura de almacenamiento que organice el espacio o dominio de búsqueda en base a cada una de las dimensiones en las que se enmarca dicho dominio.

En la búsqueda de una estructura tal, por una parte soporte básico del modelo de particionamiento para nuestra máquina QUATRO y, por otra, método eficiente de indexación de relaciones, varias alternativas fueron consideradas. El estudio realizado a tal fin perseguía el establecimiento de un método de acceso multiatributo satisfaciendo básicamente los siguientes requisitos:

- Promedio aceptable de utilización del espacio de almacenamiento, tanto en el índice como en las páginas de datos. El valor de referencia se centra en un orden de utilización promedio similar al que ofrecen métodos clásicos de acceso por clave primaria, tales como árboles B^+.
- Estructura jerárquica paginada del índice. Esta característica aporta unos rendimientos aceptables en el caso de que el índice no pueda mantenerse enteramente en memoria principal.

- Reorganización incremental sensible al crecimiento y disminución en el tamaño del fichero, evitando así el coste, en ocasiones elevado, de la reindexación.
- Habilidad para manejar búsquedas por rango, así como por coincidencia parcial y exacta.
- Poco sensible al número de atributos indexados.
- Rendimiento aceptable ante distribuciones sesgadas de datos.

Tal y como afirman Lomet y Salbergz en [44], muchos métodos de búsqueda existentes en la actualidad exhiben algunas de estas propiedades parte del tiempo. Sin embargo, la dificultad estriba en garantizar todos estos requisitos durante todo el tiempo.

La evaluación de los diversos métodos de acceso multiatributo de acuerdo a los criterios de satisfacción enunciados previamente, nos condujo a la consideración de la estructura de árbol hB como la más seria candidata como método de indexación básico para nuestra máquina QUATRO.

El árbol hB ciertamente presentaba características acorde con gran parte de nuestros objetivos. Sin embargo, un análisis exahustivo posterior de la misma reveló serias deficiencias estructurales que precisaban de una profunda reconversión de los algoritmos de manipulación del árbol. Uniendo a esto no sólo la inexistencia de algoritmos para el tratamiento de borrado en el árbol hB, sino también la dificultad añadida de divisar algoritmos de borrado que permitiesen una adaptación dinámica y coherente de la propia estructura a la situación posterior al borrado, derivamos finalmente a una solución que pasaba por la implantación de un método original que, recogiendo algunas de las inteligentes ideas planteadas por Lomet y Salbergz en [44], mostrara un comportamiento acorde con las exigencias impuestas. De este modo nació el árbol Q, cuyo nombre referencia la máquina de base de datos QUATRO, proyecto que globaliza la totalidad de nuestros estudios de investigación parte de cuyos resultados se presentan en esta tesis.

En el presente capítulo describimos pues el método de acceso indexado mediante la utilización del árbol Q. En él se reflejan los componentes, estructuras y, en general, los diferentes métodos y algoritmos necesarios para construir y utilizar el índice en árbol Q. Con objeto de presentar la estructura de árbol Q, el capítulo se organiza como sigue.

El apartado 1 describe los distintos componentes y estructuras que soportan el método de acceso del árbol Q. Esta sección se organiza en tres subapartados. El subapartado 1.1 aporta una visión global del índice en árbol Q, describiendo los componentes básicos de la estructura. En el subapartado 1.2 y con la ayuda de un ejemplo ilustrativo describimos la filosofía de construcción y crecimiento del árbol Q. En el subapartado 1.3 se definen y justifican las estructuras utilizadas en la organización de los distintos componentes del árbol.

El apartado 2 detalla los algoritmos necesarios para llevar a cabo los diferentes tipos de búsqueda en el índice.

En el apartado 3 se introduce la metodología general de inserción de datos con la utilización de un índice en árbol Q.

El apartado 4 describe en profundidad el proceso de división de nodos del árbol Q para los diferentes conjuntos que componen el árbol. Esta sección se subdivide en diferentes apartados a lo largo de los cuales se formalizan diversas propiedades de los componentes del árbol que nos permiten establecer los métodos y, en definitiva, los algoritmos necesarios para llevar a cabo la división de nodos. Con el ánimo de favorecer la comprensión de los algoritmos que se presentan, en esta sección se hace un uso extensivo de ejemplos ilustrando los diferentes casos y particularidades a tener en cuenta en el desarrollo del proceso de división.

El apartado 5 propone a modo de síntesis el algoritmo definitivo de división de nodos en el árbol Q.

Finalmente, el apartado 6 recoge las conclusiones y desarrollos futuros que se enmarcan en el establecimiento de la estructura que se presenta.

1 Descripción del árbol Q

El árbol Q es un método de acceso indexado multiatributo basado en una parcelación por niveles del dominio de búsqueda. Esta estructura se engloba, por una parte, en el conjunto de métodos de indexación multiatributo cuyo objetivo consiste en facilitar la búsqueda de registros por varios atributos y, por otra, se enmarca dentro del conjunto de estructuras equilibradas basadas en árboles binarios paginados, cuyo objetivo radica en minimizar el número de accesos a memoria secundaria durante el recorrido del índice.

1.1 Componentes del árbol Q

El árbol Q se compone básicamente de tres partes principales que constituyen una estructura piramidal. Son las siguientes:

- Conjunto de bloques **contenedores** de datos. Se compone de bloques o páginas cada una conteniendo una colección de registros. Cada contenedor representa un rectángulo multidimensional del dominio de búsqueda, es decir, una región convexa del espacio. El conjunto de bloques contenedores no es sino el fichero de datos en cuyos bloques físicos se guardan los registros. Esta parte representa la base última de la pirámide.
- Conjunto de **regiones**. Constituido por nodos que representan regiones del dominio de búsqueda. Cada región indexa un conjunto de bloques contenedores de datos. Dentro de cada región se almacena las fronteras que separan los distintos bloques

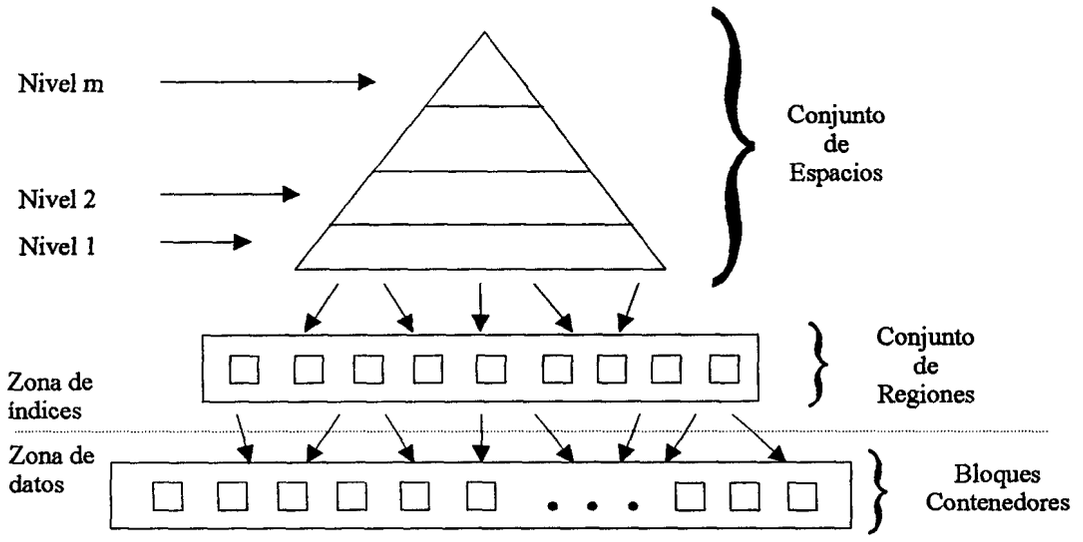


Figura 3-1. Componentes básicos de la estructura de árbol Q.

contenedores. Tras la aparición de un número de contenedores, una región puede llegar a perder la propiedad inicial de convexa y hasta dejar de representar una zona conexa del dominio de búsqueda.

- Conjunto de **espacios**. Es una pirámide de *m* niveles formada por nodos índice llamados espacios. Los nodos índice en la base de esta pirámide se encuentran a nivel 1, los inmediatamente superiores se encuentran a nivel 2, y así sucesivamente. Cada nodo de nivel *i* se denomina espacio de nivel *i*. Un espacio de nivel 1 representa una zona del dominio de búsqueda conteniendo un conjunto de regiones e indexa a dicho conjunto de regiones. Un espacio de nivel *i* ($i > 1$) indexa un conjunto de espacios de nivel $i - 1$.

Mientras que el conjunto de contenedores agrupa la **zona de datos** del árbol Q, los conjuntos de regiones y espacios constituyen la **zona de índices** del árbol.

Con objeto de utilizar una terminología homogenea en la descripción de nuestra estructura es preciso, antes de proseguir, tener en cuenta las siguientes definiciones:

- Cada nodo del árbol Q se denomina **Q-nodo**.
- Si un Q-nodo D es descendiente de algún Q-nodo de nivel superior A, decimos que A **gobierna** a D o que D es gobernado por A.
- Los Q-nodos de la zona de índices están internamente organizados en forma de árboles binarios kd. El árbol binario interior a un Q-nodo se denomina **árbol local** a dicho Q-nodo. Cada nodo del árbol local se denomina **nodo local**.

La figura 3-1 aporta una visión global del árbol Q describiendo cada uno de sus componentes.

El árbol Q, igual que el resto de estructuras índice equilibradas basadas en árboles paginados, crece desde la base de la pirámide hacia arriba.

Antes de describir las estructuras utilizadas para representar los diferentes nodos del árbol Q, veamos con la ayuda de un ejemplo gráfico, cual es la filosofía de construcción del árbol Q.

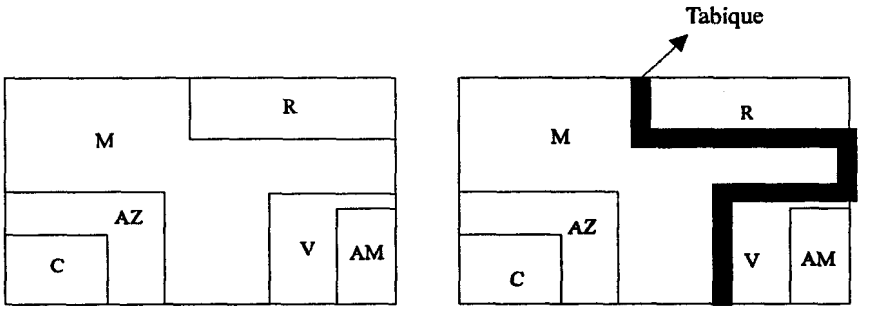
1.2 Ejemplo descriptivo del árbol Q

Supongamos que nuestro objetivo es la construcción coloreada de habitáculos de distinto nivel para llegar al diseño final de un hábitat. Así, partiendo de un suelo vacío, lo diseñamos con losetas de distinto color. Posteriormente insertamos tabiques de colores distintos y nos aparecen habitaciones, luego colocamos fachadas de color distinto para definir el dominio de una casa y así, sucesivamente, vamos definiendo el diseño global de nuestro hábitat. Conviene decir que a pesar de que el hábitat que finalmente se diseña difiere de nuestro concepto básico del término "hábitat", el ejemplo elegido se adapta convenientemente a la filosofía que pretendemos ilustrar y resulta sencillo de entender.

Inicialmente partimos de un suelo cubierto enteramente por una loseta de un determinado color. El espacio cubierto por este suelo representa nuestro dominio global de búsqueda. Cada loseta está "gobernada" por un supervisor encargado de decidir cuándo dicha loseta debe ser seccionada para utilizar un color diferente. Para realizar el trabajo, se disponen de losetas rectangulares de todos los tamaños pero de un conjunto restringido de colores.

En un momento determinado el supervisor de loseta decide cambiar de color. En este caso es preciso extraer una sección rectangular de la loseta inicial para incluir allí una nueva loseta de diferente color, apareciendo un nuevo supervisor de loseta encargado de gobernar la nueva loseta. El proceso continúa haciendo extracciones y colocando nuevas losetas de diferente color en la loseta original o incluso en losetas que ya son de color distinto al original. Conviene hacer notar que la extracción de una o varias secciones rectangulares en una loseta de un color concreto, puede "agujerear" una loseta de tal modo que varios pedazos pertenecientes a una misma loseta se encuentren distanciados entre sí. Se trataría entonces de una loseta no conexa. En cualquier caso es el color quien define el dominio de una loseta concreta, esto es, cada supervisor de loseta gestiona un color concreto y solo uno.

Para coordinar a todos los supervisores de loseta y mantener la información sobre los distintos colores que se utilizan en nuestro suelo inicial, un supervisor de mayor nivel contempla desde una posición más alta cómo se está desarrollando el diseño global de nuestro hábitat. Es el supervisor de habitación. El supervisor de habitación, en virtud de la información que maneja, conoce en todo momento cuántos colores se han utilizado en la habitación que gobierna (inicialmente existe únicamente una habitación). Cuando un determinado número de colores se utilizan dentro de una habitación, el supervisor de habitación envía una orden de saturación. En este momento se coloca un tabique de un determinado color que divide la



(a) Un supervisor de habitación

(b) Un supervisor de casa gobierna la división en habitaciones.
Dos supervisores de habitación controlan la división de habitaciones en losetas.

Leyenda: (R)ojo, (M)agenta, (AZ)ul, (C)yan, (V)erde, (AM)arillo

Figura 3-2. Suelo enlosetado con losetas de distinto color. Un supervisor de habitación controla el enlosetado. (a) Suelo antes de la división. (b) Despues de la división.

habitación original en dos, apareciendo así un nuevo supervisor de habitación. Pero ahora es necesario un supervisor de mayor nivel situado en una atalaya más alta que sea capaz de recoger la información sobre los tabiques colocados y los colores de los mismos. Se necesita pues un supervisor de casa.

En la organización jerárquica de supervisores, cada supervisor debe "rendir cuentas" ante un superior, de modo que cada supervisor es responsable en cierta medida de los supervisores encargados de gestionar las entidades de menor nivel gobernadas por aquél.

Para que nuestro sistema de diseño sea equilibrado, es preciso que todos los supervisores del mismo nivel tengan aproximadamente la misma carga de trabajo. En primer lugar, todos los supervisores tienen la misma capacidad de trabajo. Cuando esta capacidad se ve sobrepasada, el propio supervisor emite una señal de sobrecarga y se pone en marcha la maquinaria de reparto de carga. En el caso de supervisores de habitación y superiores, la carga de trabajo está definida por el número de colores que puede tratar cada supervisor. Cuando se sobrepasa el número de colores prefijado "por contrato", el supervisor emite una señal de sobrecarga[1]. Por otra parte, el sistema de reparto de carga se asegura de asignar un mínimo de carga cuando se produce el reparto. De este modo mantenemos un equilibrio de carga entre supervisores de igual nivel.

Buscando la organización jerárquica y coordinación más sencilla posible de todos los supervisores, conviene que cada supervisor de un nivel concreto rinda cuentas ante un solo supervisor de nivel superior. Así pues, en el caso de que en una habitación se hayan utilizado todos los colores posibles y sea preciso colocar un tabique para dividir la habitación, no

1 En el caso de supervisores de loseta, digamos que son otros los aspectos que definen su carga máxima de trabajo. Posteriormente entraremos a definir tales aspectos.

permitimos que una loseta de color determinado quede a ambos lados del tabique. De este modo aseguramos que el supervisor de loseta rinda cuentas ante un solo supervisor de habitación. Como puede imaginarse, debido a los tamaños irregulares que tienen las losetas de una habitación, construir un tabique que cumpla el requisito impuesto resulta en ocasiones más complejo de lo que podría parecer. Como adicionalmente es posible que existan losetas compuestas por partes no conexas, el concepto de "tabique" y por consiguiente de "habitación" en este ámbito difiere con aquél que habitualmente tenemos.

Desde una perspectiva geométrica o geográfica, las habitaciones que se construyen normalmente no son rectangulares (digamos que los tabiques se construyen con planchas que forman ángulos rectos entre sí) y, en ocasiones, el espacio que engloba una habitación no es ni siquiera conexo. Con el fin de ilustrar este punto, consideremos la figura 3-2a. Esta figura muestra el suelo de una habitación con losetas de 6 colores diferentes. Si esta habitación tuviera que ser dividida, no existe forma alguna de construir un tabique que divida esta habitación en otras dos con el mismo número de colores en cada una de ellas y abarcando cada habitación un espacio conexo. Una posible solución aparece en la figura 3-2b.

De igual forma que las losetas nuevas tienen colores diferentes, los tabiques nuevos poseen colores distintos. A medida que el trabajo avanza, es posible que un supervisor de casa quede saturado de trabajo debido a que ya se han utilizado tabiques de todos los colores disponibles para la casa cuyo diseño controla. En este caso, el supervisor de una casa mandaría una orden para construir una fachada que divida la casa, de modo que aparezca otra casa con su correspondiente supervisor. Ahora debe intervenir un supervisor de mayor nivel que gobierne cómo se han colocado las fachadas y el color de las mismas configurando las distintas casas de nuestro hábitat. Es el supervisor de ciudad.

Las casas, igualmente que las habitaciones, pueden ser no conexas, de modo que para ir de una habitación a otra de la misma casa es posible que haya que pasar por casas extrañas. Esto puede llegar a provocar con el tiempo que parte de una casa pueda ser asignada a una ciudad y otra parte de la misma casa a otra ciudad diferente, lo que se traduce en que un supervisor de casa tenga que rendir cuentas ante varios supervisores de ciudad, complicando así el trabajo coordinado de todo el equipo. Es por ello que esta situación debe ser evitada cuanto sea posible.

Es de notar que el trabajo de los supervisores es más complejo a medida que aumenta su nivel ya que, como hemos visto, bajo ciertas circunstancias un determinado supervisor de alto nivel debe encargarse de controlar el diseño en varias localizaciones diferentes (no conexas).

Los objetivos principales que se persiguen en el diseño coordinado de nuestro hábitat particular son los siguientes:

1 Todos los supervisores, salvo el de mayor nivel, están obligados a satisfacer una carga mínima de de trabajo.

2 Todos los supervisores de un mismo nivel deben tener una carga de trabajo similar.

3 Es preciso facilitar en lo posible el trabajo coordinado de supervisores de diferente nivel, de modo que, siempre que sea posible, un supervisor de nivel *i* sólo tenga que "rendir cuentas" ante un único supervisor de nivel *i+1*.

Debido a que bajo ciertas circunstancias, los objetivos 1 y 3 entran en conflicto, en la construcción de nuestro sistema priorizamos siempre el objetivo 1 antes que ningún otro. Una vez conseguido éste, atacamos el objetivo 3 y finalmente, prestamos atención al objetivo 2. Estos objetivos se concretan convenientemente en algoritmos específicos, los cuales se describirán despues de haber establecido claramente las conexiones de nuestro sistema de diseño ilustrativo con las estructuras de datos utilizadas en la organización del árbol Q. En principio podemos ya establecer que cada supervisor de un nivel determinado representa un Q-nodo, a saber:

- Supervisor de loseta: Representado por un Q-nodo del conjunto de regiones.
- Supervisor de habitación: Representado por un Q-nodo de nivel 1 del conjunto de espacios.
- Supervisor de casa: Representado por un Q-nodo de nivel 2 del conjunto de espacios.
- ...

Desde el punto de vista estructural, para asistir el trabajo de los supervisores a diferentes niveles de diseño, éstos utilizan árboles binarios que permiten describir la organización de la entidad que cada supervisor gobierna (losetas, habitaciones, casas, ciudades, etc.). En el siguiente apartado describimos con detalle las distintas estructuras utilizadas en los nodos de un árbol Q.

1.3 Estructura interna de los Q-nodos

Así como los contenedores de datos no requieren ninguna estructura específica, son sencillamente bloques conteniendo registros de datos, los Q-nodos de la zona de índices (regiones y espacios) mantienen estructuras específicas que les permiten representar las fronteras que delimitan el contenido de cada uno de ellos. Estas estructuras vienen dadas en forma de árboles k-d en donde los nodos representan fronteras del dominio de búsqueda y las hojas representan zonas del espacio.

Describimos a continuación la organización interna de los Q-nodos en los diferentes conjuntos que componen el árbol Q.

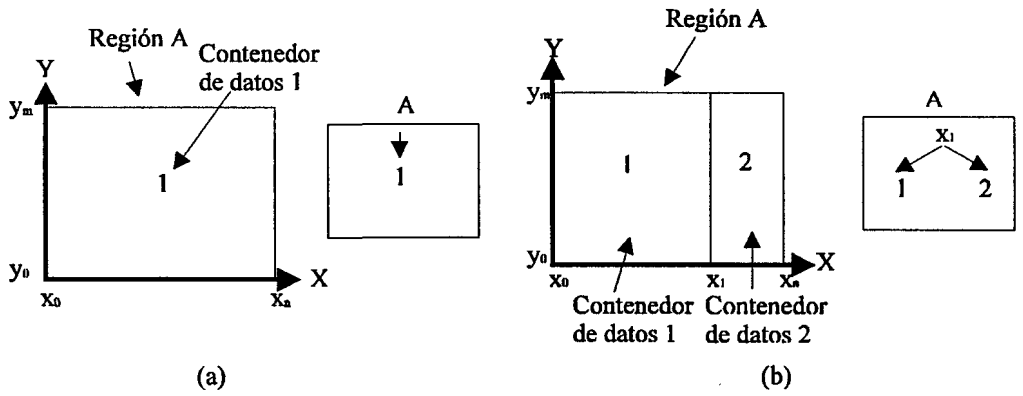


Figura 3-3. Arbol Q inicial compuesto de una sola región *A*. (a) Contenedor y región inicial antes de la división. (b) Árbol Q después de la división del contenedor 1.

1.3.1 Contenedores.

Los contenedores de datos no precisan de ninguna organización interna especial. En realidad un contenedor no es más que una página de datos donde se almacenan las tuplas a medida que éstas van llegando al fichero de datos (relación).

1.3.2 Regiones.

La estructura interna utilizada para representar una región es el árbol k-d. Según describimos en el capítulo anterior, el árbol k-d es un árbol binario en donde cada nodo almacena un valor para una dimensión del dominio de búsqueda, la dimensión a que se refiere dicho valor y dos apuntadores a hijos. El subárbol izquiedo delimita la zona del dominio de búsqueda conteniendo puntos cuyo valor para la dimensión correspondiente es menor o igual al valor representado en el nodo padre. El subárbol derecho delimita la zona del espacio conteniendo puntos cuyo valor para la dimensión correspondiente es mayor que el valor del nodo padre. Las hojas del árbol son apuntadores a los contenedores de datos. Adicionalmente, bajo determinadas circunstancias, será preciso mantener en el nodo interno del árbol k-d un valor para una dimensión extra, tal y como describiremos más adelante.

Para ver con mayor claridad cómo se organiza una región, comencemos con un dominio de búsqueda bidimensional, en el que existe una región inicial cuyo árbol k-d contiene una sola hoja (referencia al contenedor que almacena todos los puntos del espacio) como muestra la figura 3-3a. Cuando el contenedor 1 se desborda, debe ser dividido. Supongamos que utilizamos la dimensión X para realizar la división. La linea $X = x_1$ marca esta división, de modo que todos los puntos a la izquierda de esta linea son extraidos del contenedor 1 y situados en un nuevo contenedor 2. La división del espacio y el árbol k-d local a la región se muestran en la figura 3-3b. Como es lógico, la división de un contenedor debería hacerse por una linea

tal que aproximadamente la mitad de los puntos cayesen a un lado de la linea y la otra mitad al otro. Momentaneamente, suponemos que siempre es posible encontrar una linea $X = x_i$ ó $Y = y_j$ cumpliendo este requisito.

Referenciando nuestro sistema de diseño, existiría actualmente un único supervisor de loseta (Región A) asistido estructuralmente por un árbol k-d, cuyo contenido indicará a dicho supervisor cuándo éste se encuentra sobrecargado de trabajo.

A medida que nuevos puntos van llegando al dominio de búsqueda, nuevas divisiones provocan el crecimiento del árbol k-d local a la región A. Dado que el tamaño de la página física donde se almacena la región A es limitado, el número de nodos del árbol k-d local es limitado, de modo que en el momento que se alcance este límite, la Región A queda en sobrecarga. Volviendo a nuestro ejemplo de diseño, sería esta circunstancia la que provocaría que el supervisor de loseta enviara un mensaje al sistema indicando la necesidad de cambiar de color por saturación de la misma. Este aspecto de la saturación de color en el caso de una loseta, sería el que definiría la carga máxima de trabajo alcanzada por el supervisor de loseta. La figura 3-4 muestra la Región A de la figura 3-3 después de que un número de divisiones se han realizado en respuesta a la llegada de nuevas tuplas.

Desde una perspectiva genérica, la suposición anterior referente a la posibilidad de encontrar, en el ámbito de un contenedor sobrecargado, un valor para una dimensión que divida los puntos del contenedor en dos partes similarmente cargadas es demasiado restrictiva. En efecto, si consideramos por ejemplo una relación R(K,X,Y,Z) y nuestro interés se centra en organizar los datos de R en base a los atributos X e Y, puede suceder el caso de que un contenedor de datos tenga que ser dividido por sobrecarga de tuplas y una gran proporción de las tuplas referidas (o lo que es peor, todas ellas) tengan el mismo valor sobre ambos atributos X e Y. En este caso no habría forma de dividir el contenedor sobrecargado sin la participación de algún atributo adicional. Llegados a este punto no queda otro remedio que

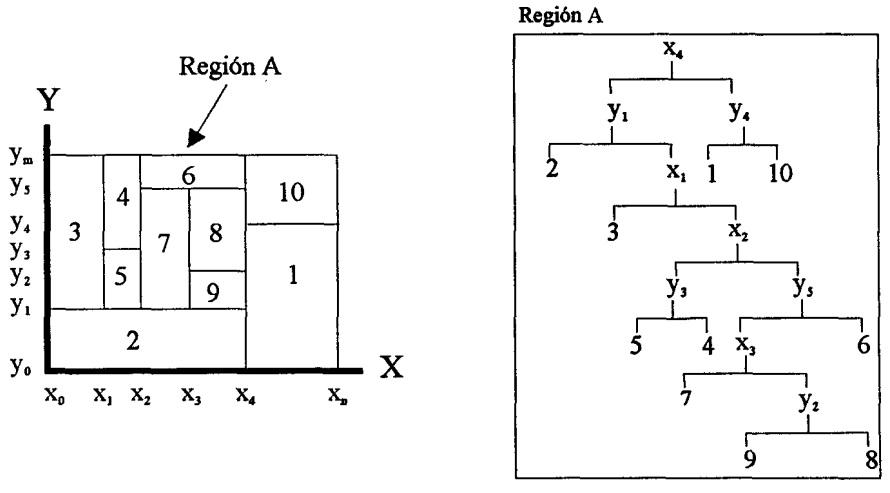


Figura 3-4. Visión geométrica de la región A y su representación mediante un árbol k-d.

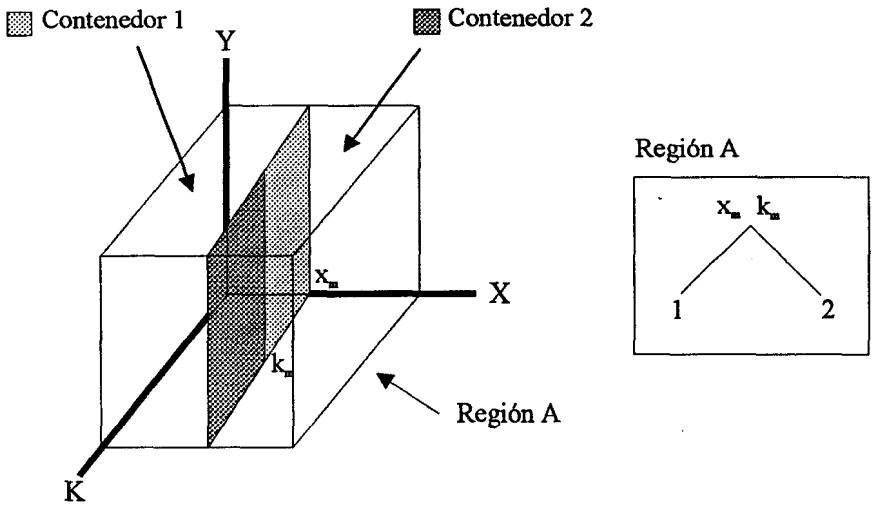


Figura 3-5. División de un contenedor en el espacio tridimensional. Muchos puntos residen en el plano $X = x_m$, por lo que se utiliza el valor de clave primaria k_m para dividir los puntos del plano entre ambos contenedores.

hacer intervenir a la clave primaria K de R.

Con la premisa de la existencia de una clave primaria para toda relación, habitualmente un atributo simple (DNI de un ciudadano, NSS de un trabajador, ISBN de un libro, etc), el sistema permite dejar a elección del usuario la utilización mínima de cada contenedor. Si ninguno de los atributos por los que estamos indexando la relación, consigue una división del contenedor garantizando la restricción de carga mínima, un valor añadido de clave primaria permitirá definitivamente efectuar una división del contenedor en dos partes con la misma carga de tuplas. Los detalles específicos sobre la forma de dividir convenientemente un contenedor se retrasarán momentaneamente.

Desde el punto de vista geométrico, el dominio de búsqueda para la relación R del ejemplo puede percibirse como un espacio tridimensional. Un contenedor de datos sería pues un cubo delimitado por los valores máximo y mínimo para los atributos X e Y. Habitualmente los puntos de un plano delimitando un cubo se encuentran totalmente dentro o fuera de él. Sin embargo, cuando no es posible conseguir el requisito de carga mínima para un cubo, partes del plano que delimita un cubo deben considerarse interiores al cubo y parte exteriores a él. La figura 3-5 muestra la organización de una región cuando es necesario utilizar la clave primaria para dividir un contenedor. Esta figura ilustra también la utilización de un valor de la clave primaria en el nodo interno del árbol k-d local a la región correspondiente.

1.3.3 Espacios.

A medida que nuevos contenedores se van adjuntado a una región determinada, el árbol k-d local a la región va creciendo hasta que se alcanza el límite de carga de la región (en nuestro ejemplo de diseño del hábitat, cuando se han utilizado losetas de todos los colores). En este caso, cuando un nuevo nodo debe ser insertado en el árbol k-d local a una región, éste

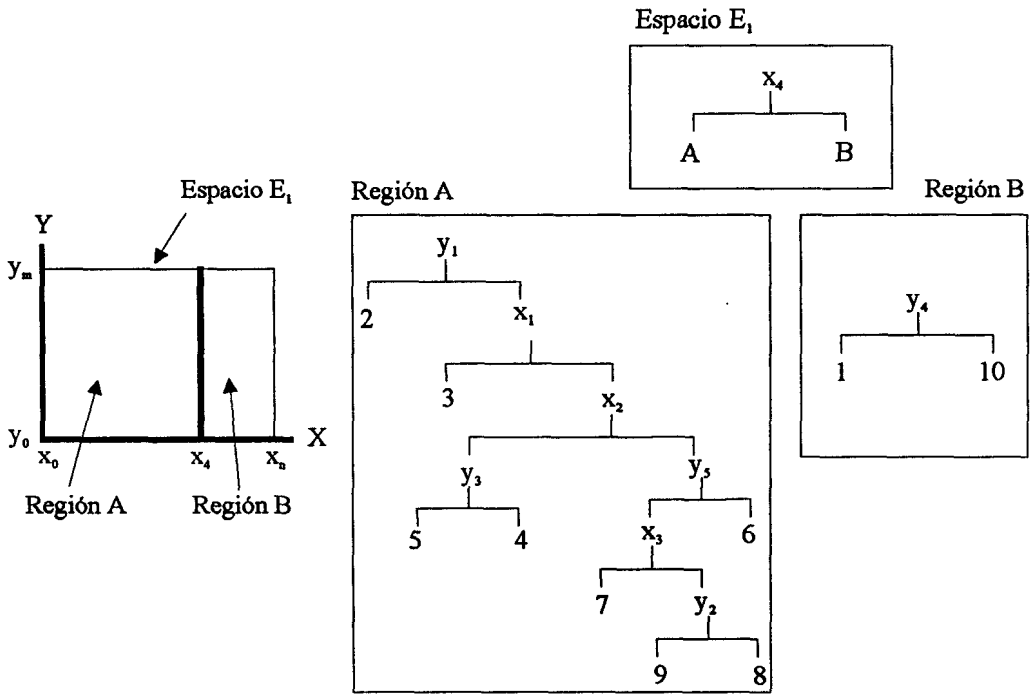


Figura 3-6. La región *A* previamente sobrecargada se divide por su antigua raíz. Como consecuencia, la carga de las dos regiones resultantes *A* y *B* no es uniforme. Tras la división aparece el espacio E_1 gobernando ambas regiones.

ya no cabe en el espacio físico destinado a la misma y se hace necesario dividir el árbol. Indudablemente, la forma más sencilla de dividir el árbol k-d local es por su raíz, pero entonces corremos el riesgo de realizar una división poco uniforme. En efecto, volviendo a la figura 3-4, si consideramos la división de la región *A* por su raíz, el contenido de las regiones resultantes aparece desproporcionado, tal y como muestra la figura 3-6.

Haciendo mención del primero de los objetivos en nuestro modelo de diseño del hábitat, según el cual "todos los supervisores, salvo el de mayor nivel, están obligados a satisfacer una carga mínima de trabajo", nos encontramos con la necesidad de repartir los contenedores que gobierna una región de tal modo que las dos regiones resultantes gobiernen ahora un mínimo de contenedores. Hablando en términos estructurales, hemos de dividir el árbol local a una región de forma que cada árbol producto de la división resulte con un mínimo número de hojas.

Desafortunadamente, dado que un árbol local a una región no tiene por qué estar equilibrado, no podemos asegurar que este mínimo esté próximo a la mitad de las hojas (lo cual conseguiría alcanzar el segundo de los objetivos propuestos). De modo general, en el peor de los casos, sólo podemos asegurar la división de un árbol binario tal que los árboles resultantes contengan entre $\lceil m/3 \rceil$ y $\lfloor 2m/3 \rfloor$ hojas, siendo m el número de hojas del árbol original. El teorema 3-1 demuestra esta afirmación.

El teorema 3-1 garantiza una utilización mínima de 1/3 de su capacidad en todas las regiones. Este valor de ocupación se designa como la carga mínima que deben satisfacer todos los Q-nodos de la zona de índices, es decir regiones y espacios. Así pues, cuando dentro de un árbol local, un subárbol satisface que su número de hojas se encuentra entre $\lceil m/3 \rceil$ y $\lfloor 2m/3 \rfloor$, siendo m el número de hojas, se dice que dicho subárbol cumple el **requisito de carga**.

Con tales premisas podemos ya establecer el modo de dividir un árbol local a una región sobrecargada. El método consiste sencillamente en extraer del árbol local un subárbol que satisfaga el requisito de carga. En la demostración del teorema 3-1 se propone implícitamente un algoritmo para encontrar dentro del árbol local un subárbol cumpliendo el requisito de carga.

Conviene observar que aunque un subárbol de una región convexa sobrecargada delimita en cualquier caso una nueva región convexa (rectangular en el caso bidimensional), el árbol resultante después de haber extraido un subárbol ya no define generalmente una región convexa, sino una que está "agujereada" (un rectángulo "agujereado" por un rectángulo interior en el caso bidimensional).

Las fronteras que definen la nueva región extraída vienen descritas de alguna forma por aquellos nodos del árbol k-d local que se encuentran en el camino desde la raíz hasta el nodo local padre del subárbol extraido.

El espacio de nivel 1 encargado de gobernar la región sobrecargada debe incluir la información necesaria para discernir el contenido de las dos nuevas regiones que aparecen como producto de la división. Esta información no es otra que la frontera que delimita el

Teorema 3-1. *División de un árbol binario con ratio garantizado de 2:1.*

En un árbol binario con m hojas, donde cada nodo interno posee necesariamente dos hijos, siempre es posible encontrar un subárbol conteniendo un número de hojas mayor o igual que $\lceil m/3 \rceil$ y menor o igual que $\lfloor 2m/3 \rfloor$.

Demostración.

Partiendo de la raíz, si alguno de sus hijos cumple la condición, el teorema queda demostrado. En caso contrario, alguno de ellos posee un número de hojas mayor que $\lfloor 2m/3 \rfloor$. En efecto, llamemos r a la raíz del árbol original y sean c_1 y c_2 las raices de los subárboles descendientes de r. Se tiene que:

$$m = \text{hojas}(r) = \text{hojas}(c_1) + \text{hojas}(c_2)$$

Si $\text{hojas}(c_1) \leq \lfloor 2m/3 \rfloor$ y $\text{hojas}(c_2) \leq \lfloor 2m/3 \rfloor$, entonces, puesto que ninguno de los dos subárboles satisface la condición, tenemos que $\text{hojas}(c_1) < \lceil m/3 \rceil$ y $\text{hojas}(c_2) < \lceil m/3 \rceil$. Por consiguiente

$$m = \text{hojas}(c_1) + \text{hojas}(c_2) < \lceil m/3 \rceil + \lceil m/3 \rceil = 2\lceil m/3 \rceil \quad (1)$$

Por su parte, dado m exite un número natural $k \geq 0$ tal que $m = 3k$, ó $m = 3k + 1$, ó $m = 3k + 2$.

$\Rightarrow$

...

Si $m = 3k$, entonces por (1) tenemos

$$m = 3k < 2\lceil m/3 \rceil = 2\lceil 3k/3 \rceil = 2k$$

lo cual es una contradicción.

Si $m = 3k + 1$, entonces

$$m = 3k + 1 < 2\lceil (3k+1)/3 \rceil = 2(k + 1) \Rightarrow k < 1 \Rightarrow k = 0$$

Pero entonces estamos considerando un árbol con $m = 1$ hoja y por tanto sin descendientes.

Si $m = 3k + 2$, entonces,

$$m = 3k + 2 < 2\lceil (3k+2)/3 \rceil = 2(k + 1) \Rightarrow k < 0$$

lo cual es una contradicción.

Lo anterior demuestra que si no hay hijos satisfaciendo la condición propuesta, alguno de ellos será la raíz de un árbol conteniendo más de $\lfloor 2m/3 \rfloor$ hojas. Con estas premisas continuamos descendiendo el árbol por el subárbol hijo que más hojas contenga. Después de un número finito de pasos llegaremos a un subárbol tal que ninguno de los dos hijos contenga más de $\lfloor 2m/3 \rfloor$ hojas. Sea x la raíz de dicho subárbol y sean c_1 y c_2 las raices de los subárboles descendientes de x. Si alguno de los subárboles con raices c_1 o c_2 contiene más de $\lceil m/3 \rceil$ hojas, el teorema queda demostrado. En caso contrario tenemos:

$$\text{hojas}(c_1) < \lceil m/3 \rceil \text{ y } \text{hojas}(c_2) < \lceil m/3 \rceil.$$

Y por tanto

$$\lfloor 2m/3 \rfloor < \text{hojas}(x) = \text{hojas}(c_1) + \text{hojas}(c_2) < \lceil m/3 \rceil + \lceil m/3 \rceil = 2\lceil m/3 \rceil$$

Si $m = 3k$, entonces

$$2k = \lfloor 2m/3 \rfloor < 2\lceil m/3 \rceil = 2k.$$

Si $m = 3k + 1$

$$2k = \lfloor 2(3k + 1)/3 \rfloor = \lfloor 2m/3 \rfloor < \text{hojas}(x) < 2\lfloor m/3 \rfloor = 2(k + 1) \Rightarrow \text{hojas}(x) = 2k + 1$$

Pero

$$\text{hojas}(c_i) < \lceil m/3 \rceil = k + 1 \ (i = 1,2) \Rightarrow \text{hojas}(c_i) \leq k \ (i = 1,2),$$

luego

$$2k + 1 = \text{hojas}(x) = \text{hojas}(c_1) + \text{hojas}(c_2) \leq 2k$$

lo que nos lleva a un absurdo.

Finalmente si $m = 3k + 2$

$$2k + 1 = \lfloor 2(3k + 2)/3 \rfloor = \lfloor 2m/3 \rfloor < \text{hojas}(x) = \text{hojas}(c_1) + \text{hojas}(c_2) < 2\lceil m/3 \rceil = 2(k + 1)$$

lo cual es un absurdo ya que $2k + 1$ y $2k + 2$ son números naturales consecutivos y hojas(x) es también un número natural. ♦

contenido de ambas regiones.

Con objeto de valorar las posibles estructuras que permitan representar la información local a los espacios, conviene resaltar las siguientes apreciaciones:

- Un árbol k-d que integre el camino completo desde la raíz del árbol local a la región sobrecargada hasta el nodo local padre del subárbol extraído, permite representar con toda fidelidad la frontera que separa las dos regiones que aparecen tras la división. Sin embargo, este árbol k-d mantendrá en sus hojas referencias duplicadas hacia la antigua región sobrecargada. La figura 3-7a ilustra este hecho. Como puede verse en la misma, la región A, agujereada tras la extracción de B, aparece referenciada varias veces en el árbol local a E_1.
- La altura de un árbol k-d que represente la frontera tras la división, dependerá en principio de la profundidad a la que se sitúe el árbol extraído dentro del árbol local a la región sobrecargada. En casos extremos, la representación de la frontera en forma de árbol k-d podría precisar de un número importante de nodos y, en consecuencia, de una cantidad elevada de referencias redundantes.
- El camino que define la frontera de la región extraída puede contener nodos redundantes. En efecto, de todos aquellos nodos locales conteniendo valores para la misma dimensión tales que para alcanzar el subárbol extraído sea preciso bajar por una rama situada en la misma dirección, sólo el que se localiza a mayor profundidad es necesario para delimitar la región extraída. El resto resultan ser nodos redundantes.

Esta última consideración merece clarificarse con la ajuda de un ejemplo. Volviendo sobre la figura 3-7a, dado que la región extraída B se sitúa a la derecha de la linea $X = x_2$, los puntos del dominio de búsqueda pertenecientes a contenedores gobernadas por B, son tales que sobre su dimensión X, poseen valores mayores que x_2. Resulta entonces inútil incluir como información de frontera (en el espacio E_1 de la figura, esta información la constituyen todos los nodos que nos conducen a B) que los puntos gobernados finalmente por B tienen en su dimensión X un valor mayor que x_1, ya que $x_1 < x_2$. Esto significa que, en el peor de los casos, considerando un dominio de *k* dimensiones, un máximo de $2k$ valores serían suficientes para representar la frontera que delimita la región extraída. Estos $2k$ valores no son otros que las cotas superior e inferior de los rangos que definen la región extraida para cada una de las dimensiones del dominio de búsqueda.

Las anteriores apreciaciones nos llevan a considerar dos posibles alternativas estructurales para organizar regiones:

1 Un árbol k-d que incluya el camino definiendo la frontera que separa regiones resultantes tras la división.

2 Un árbol binario con nodos locales multidimensionales, es decir cuyos nodos internos incluyan los rangos para cada dimensión que delimitan la región extraida. En este caso, el hijo izquierdo del nodo representa la zona interior a la región extraida mientras que el hijo derecho representa la zona exterior a dicha región. La figura 3-7b muestra el espacio E_1 organizado ahora mediante un árbol binario como el descrito.

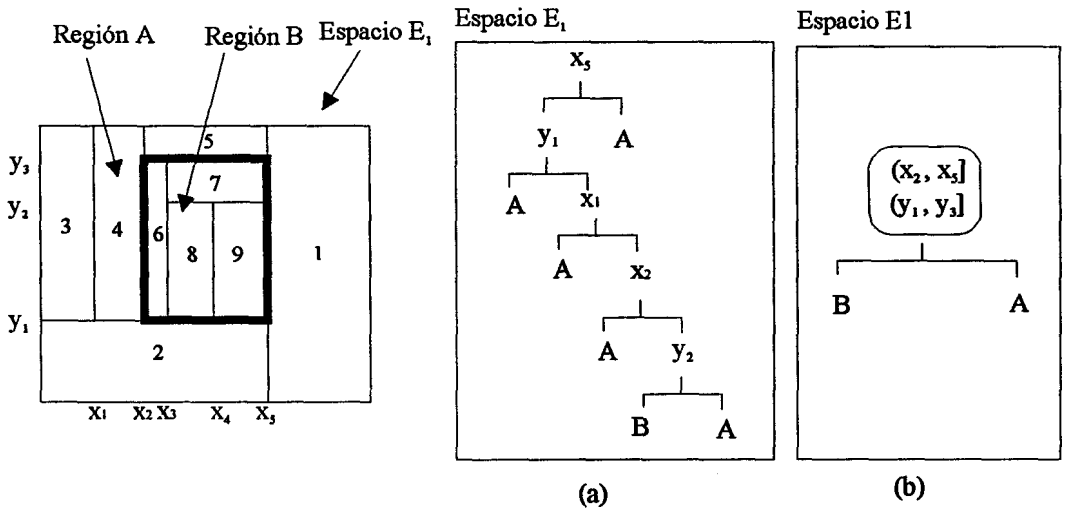


Figura 3-7. Dos posibles representaciones de una región. (a) Representación por medio de un árbol k-d. (b) Representación a través de un árbol binario cuyos nodos locales almacenan las fronteras de una región conexa. El hijo izquierdo representa al interior de la región y el derecho al exterior de la misma.

Ambas alternativas acarrean ventajas e inconvenientes. Por una parte, la utilización de un árbol k-d aporta homogenidad en el tratamiento de Q-nodos de la zona de índices del árbol Q (todos los Q-nodos pueden representarse mediante árboles k-d). Los algoritmos de división de Q-nodos y traslado de nodos locales no precisan tratar con estructuras diferentes en función del conjunto del nivel al que pertenezcan dichos Q-nodos. Por otra parte y como ya hemos mencionado, utilizar un árbol k-d como estructura local para los espacios supone asumir una cierta cantidad de redundancia, tanto en sus hojas como en sus nodos internos. Como veremos más adelante la existencia de referencias redundantes conducen, a menos que se les preste la debida atención, a serios problemas debido a la potencial aparición de Q-nodos multipadre, es decir Q-nodos gobernados por distintos nodos del nivel superior.

Por su parte, la utilización de árboles binarios con nodos multidimensionales evita en el nivel 1 del conjunto de espacios la aparición de referencias redundantes a las regiones gobernadas. Sin embargo esto no significa la eliminación definitiva de esta redundancia en niveles superiores del conjunto de espacios. Además, en el caso de que el número de dimensiones sea elevado, el coste de almacenamiento de un nodo local puede resultar excesivo. Teniendo en cuenta que el valor de clave primaria es susceptible de ser utilizado en la división de contenedores, las cotas inferior y superior de cada uno de los intervalos representados en un nodo local k-dimensional pueden llevar adjunto un valor de clave primaria. Siendo así, en el peor caso, $4k$ valores deben ser incluidos en un nodo multidimensional, agravando considerablemente el coste en espacio.

Las valoraciones anteriores, unido a los análisis que se incluyen en [44] sobre las ventajas en los costes de espacio y tiempo que proporciona el uso de árboles k-d frente a la utilización de información de frontera en forma de rangos, nos conducen finalmente a considerar el árbol k-d como candidato idoneo para organizar el contenido de los Q-nodos del conjunto de espacios.

Los estudios que aparecen en [44] no consideran, sin embargo, algunas de las ventajas aquí expuestas sobre el uso de árboles binarios con nodos locales k-dimensionales frente a los árboles k-d. La implementación del árbol Q mediante el uso de árboles binarios con nodos k-dimensionales es objeto de ampliación futura de la presente tesis. El análisis y comparación de las distintas alternativas de representación del árbol Q puede aportar resultados significativos para valorar en qué situaciones puede resultar ventajoso utilizar una u otra de las estrategias.

Habiendo establecido definitivamente la estructura interna de los Q-nodos índice como árboles k-d, nos encontramos ante el reto de disminuir en lo posible las consecuencias negativas que conlleva nuestra decisión. Dos cuestiones se plantean de inmediato:

- ¿Podríamos evitar la inclusión de todos y cada uno de los nodos locales en el camino desde la raíz del árbol local a la región sobrecargada hasta el padre del árbol extraído, restringiendo en el peor caso el número de nodos en dicho camino a $2k$?.
- Asumiendo el riesgo que conlleva la aparición de referencias redundantes en las hojas del árbol k-d local a un Q-nodo, ¿cómo disminuir al mínimo el número de referencias redundantes a la región sobrecargada tras haberle extraido una nueva región?.

Debido a las posteriores consecuencias que acarrea la eliminación de nodos redundantes en el camino que define la nueva frontera, la respuesta a la primera pregunta es definitivamente *no*. De hecho, en [44] esta eliminación de nodos redundantes se plantea como mecanismo básico de traslado de nodos locales al Q-nodo padre de la región sobrecargada. Sin embargo, cuando producto de posteriores divisiones de la antigua región sobrecargada, se hace necesario trasladar esos nodos que fueron previamente redundantes, el algoritmo de traslado se complica en exceso, ya que es necesario situar estos nodos por encima de otros nodos locales. Esta decisión de no trasladar inicialmente nodos redundantes en el camino hacia el árbol extraído, puede conducir a un escenario erroneo que dirija la búsqueda de puntos hacia zonas incorrectas del dominio de búsqueda, tal y como se ilustra en [22]. Así pues, con objeto de simplificar, por una parte, el algoritmo de traslado de nodos y de evitar, por otra, el desastre que puede suponer un método de indexación incorrecto, en nuestros algoritmos de traslado de nodos, el camino completo que define la frontera tras la extracción es incluido como parte del árbol k-d que gobierna la región dividida.

La segunda cuestión de cómo disminuir el número de referencias redundantes, se resuelve de forma sencilla eligiendo un subárbol candidato para ser extraído cuya raíz se sitúe lo más cerca posible a la raíz del árbol local a la región sobrecargada. Esta elección incide no sólo

en una disminución de referencias redundantes, sino también en definir una frontera incluyendo un número reducido de nodos locales. Lógicamente, la restricción de carga mínima para la región extraida será en último lugar quien dicte la longitud del camino que será necesario trasladar al Q-nodo padre.

Una vez definidas las estructuras que utilizaremos para representar la información interna a los nodos del árbol Q y antes de describir en detalle los algoritmos de división y traslado de nodos locales, enunciamos en el siguiente apartado los métodos para efectuar búsquedas en el árbol Q.

2. Búsqueda en el árbol Q

Toda estructura multidimensional eficiente debe dar respuesta eficaz a los distintos tipos de consulta que soliciten datos posiblemente existentes en el dominio de búsqueda. Los distintos tipos de consulta que pueden efectuarse son:

- Busqueda **exacta:** Se solicita la localización de una tupla cuyos valores para cada dimensión aparecen en la petición. Así pues, la petición consiste en una k-tupla (v_1, v_2, ..., v_k), donde v_i es un valor del dominio del atributo correspondiente a la dimensión *i*. La búsqueda finaliza encontrando un contenedor donde, posiblemente, se localiza la tupla requerida.
- Búsqueda **parcial:** En este caso, la solicitud especifica un conjunto parcial de valores (v_{i1}, ..., v_{is}), donde $s < k$ y cada v_{ij} es un valor del dominio del atributo correspondiente a la dimensión i_j. La búsqueda consiste en localizar finalmente un conjunto de contenedores donde, si existen, se encuentran todas las tuplas cuyas componentes coinciden con aquellas que se solicitaron.
- Búsqueda **en rango**. Para este tipo de consultas se especifican *s* rangos (R_{i1}, ..., R_{is}) correspondientes a otras tantas dimensiones del dominio de búsqueda. Cada rango es un intervalo definido por sus cotas inferior y superior para cada una de las correspondientes dimensiones. Ambas cotas del rango R_{ij} son valores del dominio del atributo correspondiente a la dimensión i_j. La búsqueda finaliza localizando un conjunto de contenedores conteniendo, en caso de que existan, tuplas cuyos valores para los atributos correspondientes a las dimensiones i_j ($1 \leq j \leq s$) se encuentran dentro de los rangos solicitados.

2.1 Búsqueda exacta

La petición de una tupla con valores (v_1, v_2, ..., v_k), comienza, igual que el resto de las búsquedas, por la raíz del árbol Q. Dentro de cada Q-nodo se recorre su árbol local, confrontando el predicado de la búsqueda con el contenido del nodo local. Como sabemos, cada nodo local incluye un valor para una dimensión determinada y, eventualmente, un valor

de clave primaria. Si el valor buscado sobre la correspondiente dimensión es menor o igual que el contenido del nodo local, bajamos al hijo izquierdo. Si es mayor estricto que éste bajamos por el hijo derecho.

El punto más crítico de la búsqueda, cualquiera que sea el tipo de búsqueda que se efectúe, es aquél en que el nodo local incluye un valor adicional de clave primaria (*k*) junto con el valor para la dimensión que corresponda y, al mismo tiempo, el valor buscado v_i coincide con el valor del nodo. En este caso, es preciso determinar cómo es la tupla buscada respecto de *k*. Si no poseemos información de búsqueda sobre el valor deseado de clave primaria, es necesario bajar por ambos hijos. En otro caso podemos identificar plenamente si la tupla buscada se encuentra a un lado u otro del hiperplano que representa *k*, de modo que si es menor o igual bajamos por el hijo izquierdo y si es mayor por el hijo derecho.

El algoritmo 3-1 detalla el proceso que debe seguirse para efectuar una búsqueda exacta dentro de un Q-nodo. Este algoritmo indica por dónde debe proceder la búsqueda. En caso de que el nodo local actual sea una hoja apuntando a un nuevo Q-nodo, la siguiente aplicación del algoritmo comenzaría en la raíz del árbol local a dicho Q-nodo.

2.2 Búsqueda parcial

A diferencia de la búsqueda exacta, en un proceso de búsqueda parcial podemos encontrarnos

Algoritmo 3-1. Búsqueda exacta para Q-nodos.

Entrada: Vector de búsqueda (v_1, v_2, ..., v_k). Valor de clave primaria de la tupla buscada kpb. Si es desconocido kpb = Nulo. La comparación kbp ≤ +∞ es siempre *Verdadero* aún siendo kbp nulo. La comparación *Nulo* ≤ kp, siendo kp ≠ +∞, es siempre *Falso*.

Consideraciones: Nos encontramos en un nodo concreto del árbol local a un Q-nodo específico del árbol Q. Este algoritmo nos dice simplemente por dónde ha de continuar la búsqueda. El nodo local actual contiene un valor c_i para una dimensión concreta. El valor de clave primaria unido a c_i se denota por kp. Este valor puede ser del dominio activo o, en caso de no existir, +∞.

```
1.  si vi < ci
2.      Bajar por el hijo izquierdo
3.  si no si vi > ci
4.      Bajar por el hijo derecho
5.  si no {vi = ci}
6.      si kpb ≤ kp
7.          Bajar por el hijo izquierdo
8.      si no si kpb > kp
9.          Bajar por el hijo derecho
10.     si no
11.         Bajar por ambos hijos
12.     fin si
13. fin si
14. Fin.
```

ante un nodo del árbol local cuya dimensión no se corresponda con ninguna de las dimensiones que aparecen en el vector de búsqueda. En tal caso es necesario bajar por ambos hijos. En el resto de los casos este algoritmo es idéntico al correspondiente algoritmo de búsqueda exacta, es por ello que omitimos la descripción del mismo.

2.3 Búsqueda en rango

Como ya mencionamos en la introducción, la búsqueda en rango consiste en localizar tuplas cuyas coordenadas se encuentren dentro de los rangos especificados en el vector de búsqueda (R_{i1}, ..., R_{is}) (todos los rangos se consideran cerrados por ambos extremos). En este caso simplificamos el algoritmo extendiendo el vector de búsqueda a k rangos, donde $R_{ij} = (-\infty , +\infty)$ para j = s+1..k. En la visita a un nodo local, confrontamos su valor con el rango de

Algoritmo 3-2. Búsqueda en rango para Q-nodos.

Entrada: Vector de búsqueda (RB_1, RB_2, ..., RB_k). Rango de clave primaria deseado la tupla buscada [kpb.ci, kpb.cs]. Si es desconocido kpb.ci = -∞ y kpb.cs = +∞.

Consideraciones: Nos encontramos en un nodo concreto del árbol local a un Q-nodo específico del árbol Q. Este algoritmo nos dice simplemente por dónde ha de continuar la búsqueda. El nodo local actual contiene un valor v_i correspondiente a la dimensión i. El valor de clave primaria unido a v_i se denota por kp. Este valor pueden ser del dominio activo o ser +∞. Las cotas superior e inferior de un rango RB se denotan respectivamente por RB.ci y RB.cs.

```
1.  si RBi.cs < vi
2.      Bajar por el hijo izquierdo
3.  si no si RBi.ci > vi
4.      Bajar por el hijo derecho
5.  si no si RBi.ci > vi Y RBi.cs < vi
6.      Bajar por ambos hijos
7.  si no si RBi.ci = vi
8.      si kpb.cs ≤ kp Y RBi.ci = RBi.cs
9.          Bajar por el hijo izquierdo
10.     si no si kpb.ci > kp
11.         Bajar por el hijo derecho
12.     si no
13.         Bajar por ambos hijos
14.     fin si
15. si no si RBi.cs = vi
16.     si kpb.ci > kp Y RBi.ci = RBi.cs
17.         Bajar por el hijo derecho
18.     si no si kpb.cs ≤ kp
19.         Bajar por el hijo izquierdo
20.     si no
21.         Bajar por ambos hijos
22.     fin si
23. fin si
24. Fin.
```

búsqueda para la correspondiente dimensión. Si el valor del nodo es interior al rango, hay que bajar por ambos hijos. Si es exterior, sólo uno de los hijos debe ser visitado.

De modo similar a como sucede en el caso de las búsquedas previas, los puntos críticos del algoritmo se centran en los casos en que los extremos de los rangos de búsqueda coinciden con el valor contenido en el nodo local, en cuyo caso habrá que tener en cuenta tanto si el nodo local incluye un valor de clave primaria como si es conocido el rango deseado sobre la clave primaria.

El algoritmo 3-2 muestra los describe en detalle los pasos precisos a efectuar en el caso de una búsqueda por rango. Igual que antes, el algoritmo nos indica por qué rama dentro del árbol local al Q-nodo debe proceder el algoritmo en su siguiente aplicación.

Los algoritmos anteriores, recursivamente aplicados en cada paso del recorrido por los árboles locales a los Q-nodos, nos llevan finalmente a un conjunto de contenedores suceptibles de almacenar tuplas que satisfacen el predicado. Cada tupla de los contenedores resultantes de la búsqueda debe ser finalmente examinada para comprobar si su contenido se ajusta definitivamente al predicado de la consulta.

3. Inserción en árboles Q

Como sucede en la mayoría de estructuras de indexación basadas en árboles paginados, el algoritmo de inserción de datos en el árbol Q procede, a partir de las hojas, siguiendo una ruta ascendente. Esta filosofía de inserción en estructuras arboreas, cuya idea fue originalmente adoptada por Bayer y McCreight en sus árboles B [7], ha llegado a imponerse como mecanismo universal de inserción en tales estructuras de información. En este sentido, el algoritmo 3-3 de inserción en árboles Q no difiere en esencia de los algoritmos clásicos de inserción de datos en árboles B^+.

Las características propias y diferenciadoras de la estructura de árbol Q son las que, en último término, dictarán la forma en que el algoritmo 3-3 debe ser desarrollado. El siguiente apartado tiene como fin describir el desarrollo específico del algoritmo 3-3 sobre el árbol Q. El establecimiento formal de las propiedades del árbol Q nos permitirá justificar, a lo largo del próximo apartado, las decisiones adoptadas en la aplicación de la metodología general que describe el algoritmo 3-3 a la estructura de árbol Q que aquí se presenta.

4. División de nodos en el árbol Q

Cuando alguno de los nodos componentes del árbol Q queda sobrecargado, debe ser dividido y la información que describe esta división debe ser trasladada al Q-nodo padre. En apartados anteriores hemos adelantado parte de este proceso, sin embargo, tal y como explicamos en el

Algoritmo 3-3. Algoritmo general de inserción

Entrada: Registro a ser insertado.

1. Buscar en el Árbol Q el contenedor que contendrá el registro a ser insertado.
2. Dentro de este contenedor, encontrar la localización donde el nuevo registro será situado. Ahora dos casos pueden aparecer:
3. Si hay espacio para insertar este nuevo registro. Insertar el registro. **Fin.**
4. Si no hay espacio para insertar el nuevo registro. En este caso, el contenedor de datos debe dividirse.
5. Adjuntar un nuevo contenedor y repartir el contenido del contenedor original entre éste y el nuevo contenedor tan uniformemente como sea posible.
6. Trasladar un término índice que define la frontera entre ambos contenedores al nivel inmediatamente superior en el Árbol Q. Si el Q-nodo destino de este término índice quedara sobrecargado, dividir el Q-nodo de forma análoga. Repetir la operación mientras el Q-nodo destino del término índice quede sobrecargado.

ejemplo ilustrativo de diseño del hábitat, el trabajo de los supervisores se complica a medida que subimos de nivel. Esto se traduce en una complicación del algoritmo de traslado de nodos locales a partir de regiones hacia arriba, ya que, con objeto de alcanzar los objetivos marcados, información puntual sobre el estado de los nodos debe ser convenientemente trasladada hacia los niveles superiores. Pero vayamos paso a paso en la descripción de los algoritmos de división y traslado de nodos locales. Comenzamos por la división de contenedores.

4.1 División de contenedores

Cuando un contenedor de datos está lleno y llega una nueva tupla a dicho contenedor, éste debe ser dividido. Para realizar la división, se elige una dimensión de aquellas que componen el dominio de búsqueda para distribuir las tuplas según los valores de esta dimensión. Existen diferentes alternativas a la hora de decidir sobre qué dimensión debe efectuarse la división del contenedor. En el capítulo 5 se describen las diferentes políticas de selección de discriminadores que hemos implementado en nuestros análisis y los resultados de rendimiento obtenidos para las diferentes alternativas.

Una vez decidida la dimensión (atributo) que utilizaremos en la división del contenedor sobrecargado, ordenamos el contenedor según ese atributo y obtenemos la mediana de los valores sobre dicho atributo. Si el número de tuplas cuyo valor sobre dicho atributo es menor o igual a la mediana obtenida, satisface la restricción de carga definida por el usuario, este valor marcará la división de tuplas. En caso contrario, se obtiene además el valor mediana sobre la clave primaria de todas aquellas tuplas que, sobre la dimensión elegida, poseen el valor mediana definido antes. En este caso, ambos valores mediana definen el punto de división de las tuplas del contenedor.

Definido el punto de división (elemento discriminador), asignamos un nuevo Q-nodo al conjunto de contenedores. Las tuplas cuyos valores sobre la dimensión elegida sean estrictamente mayores que el elemento discriminador, son movidas al nuevo contenedor, permaneciendo el resto en el antiguo contenedor. Tras la división, el elemento discriminador debe ser trasladado al Q-nodo padre, es decir, a la región que gobierna el contenedor sobrecargado. En concreto, dentro del árbol local a esta región, un nuevo nodo local (término índice) debe reflejar la división que acabe de efectuarse. El lugar exacto donde debe ser colocado el término índice, es justo a partir de la hoja que apuntaba al contenedor que acabamos de dividir. Esta hoja puede localizarse durante el recorrido previo del árbol Q que es preciso realizar para encontrar el contenedor destino de la nueva tupla. El algoritmo de traslado de este término índice consiste sencillamente en reemplazar la antigua hoja por un nodo interno conteniendo el elemento discriminador. Como hijo izquierdo de este nuevo nodo se coloca la antigua referencia al contenedor sobrecargado y como hijo derecho se coloca una referencia al contenedor recién insertado en el conjunto.

4.2 División de regiones

Cuando un nuevo término índice debe ser incluido en la región como resultado de la división de un contenedor y el árbol local a esta región ocupa todo el espacio físico destinado a la misma, ésta entra en situación de sobrecarga y tiene que ser dividida. En este caso,tal y como ya hemos apuntado, la división de una región consiste en extraer del árbol local un subárbol cumpliendo ciertos requisitos.

En todo caso, cualquiera que sea el Q-nodo de la zona de índices que va a ser dividido, ya sea región o espacio, varios objetivos deben ser considerados en la elección del subárbol para ser extraido:

- Satisfacer el requisito de carga. Esto es, el árbol extraido debe contener entre un tercio y los dos tercios del número de hojas que contiene el árbol local a la región sobrecargada.
- Realizar un reparto equilibrado. Es decir, el árbol extraido debería contener un número de hojas próximo a la mitad de las hojas que contiene el árbol local a la región sobrecargada. Con esto se conseguiría un reparto de carga uniforme entre las regiones resultantes, de manera que ambas gobernarían un número similar de contenedores.
- Disminuir en lo posible el número de referencias redundantes en el Q-nodo padre. Este Q-nodo, que gobernaba la región sobrecargada, pasa ahora a gobernar las dos regiones resultantes tras la división.
- Evitar la aparición de Q-nodos multipadre. Resulta importante, como posteriormente veremos, evitar que la división de un Q-nodo provoque que algún Q-nodo gobernado por éste antes de la división no quede, tras la división, gobernado por los dos Q-nodos resultantes.

Definición 3-1. Factor de carga diferencial (FCD)

Sea A un árbol local a un Q-nodo y sea S un subárbol cualquiera de A. Se define el Factor de Carga Diferencial (FCD) de *S* como:

$$FCD(S) = | 0.5 - hojas(S)/hojas(A) |$$

En el caso de división de regiones, el último de los objetivos propuestos no tiene incidencia alguna, ya que el árbol local a una región nunca contiene en sus hojas referencias redundantes a los contenedores que gobierna. Es por ello que en este apartado, sólo prestamos atención a los tres primeros objetivos.

En anteriores apartados establecimos como prioritario el objetivo de ocupación mínima para los nodos del árbol Q. Así pues, el resto de objetivos planteados quedan sometidos a la consecución del requisito de carga. A partir de aquí podemos entrar en la discusión acerca de cual de los restantes objetivos debe ser atacado antes que el resto. Si priorizamos el reparto equilibrado, desde luego aseguramos la mejor ocupación posible de los Q-nodos, sin embargo de nada sirve mantener un nivel alto de ocupación si ello supone como contraprestación almacenar información redundante y por tanto en algún sentido estéril.

Teniendo en cuenta lo anterior y asumiendo momentáneamente el costo en complejidad algorítmica que suponen las referencias redundantes, en el caso que nos ocupa de división de regiones, el algoritmo de extracción del subárbol debe seleccionar el subárbol más cercano a la raíz del árbol local a la región sobrecargada que satisfaga el requisito de carga. Si hubiera dos, seleccionamos aquél cuyo número de hojas esté más próximo a la mitad de las hojas que incluye el árbol local original. Con tal motivo la definición 3-1 introduce el término **Factor de Carga Diferencial**.

De acuerdo con la definición 3-1, mientras menor sea el factor de carga diferencial de un subárbol, más se aproxima el número de hojas que contiene a la mitad de las hojas del árbol local a un Q-nodo. Con ayuda de la anterior definición estamos ya en disposición de definir el algoritmo de extracción de subárbol para regiones. El algoritmo 3-4 de extracción de subárbol a partir de una región sobrecargada, obliga al cumplimiento de requisito de carga de la región extraida, lo que supone un primer paso en la consecución de factores de ocupación de páginas índice que pueden considerarse aceptables. Teniendo en cuenta la necesidad de alcanzar objetivos adicionales al requisito de carga, este algoritmo se verá posteriormente actualizado con tal motivo. Pero, vayamos por partes.

Una vez extraido el subárbol e insertado en un nuevo Q-nodo añadido a tal efecto, igual que hicimos con la división de contenedores, es preciso realizar los ajustes necesarios en el árbol Q para representar la división. Estos ajustes pueden resumirse en las siguientes tareas:

1. Seleccionar el subárbol a extraer del árbol local a la región sobrecargada.

2. Adjuntar una nueva región al árbol Q. El árbol local a esta nueva región es el árbol seleccionado en el punto 1.

3. Trasladar al Q-nodo padre la frontera que delimita la región extraida. Como ya hemos visto, esta frontera está constituida por los nodos del árbol local a la región sobrecargada que se sitúan entre la raíz de dicho árbol y el padre del subárbol extraido. Como es lógico, para efectuar correctamente este traslado es preciso:

   3.a Localizar el Q-nodo padre que gobierna la región sobrecargada (suponemos inicialmente que es único).

   3.b Identificar el punto dentro del árbol local al Q-nodo padre a partir del cual hemos de insertar los nodos trasladados desde abajo.

4. Modificar el árbol original, eliminando de él el subárbol extraido junto con el nodo local padre del mismo y colocando en su lugar el subárbol hermano del subárbol extraido.

Fijémonos que una vez desplazado del árbol original el subárbol extraido, caso de no ser eliminado, el nodo local padre de este subárbol tendría ahora como uno de sus hijos algún tipo de referencia externa. De hecho ésta es la representación utilizada en los árboles hB, donde se coloca una marca externa en el lugar donde se situaba el subárbol extraido. Pero esta marca externa además de ocupar espacio en la página, se muestra completamente inútil en la búsqueda de puntos. En efecto, dado que ese nodo local padre del subárbol extraido es trasladado al Q-nodo padre, si, como resultado de una búsqueda, llegamos hasta el Q-nodo dividido y dentro de él hasta el nodo local padre del subárbol extraido, la comparación con el valor buscado nunca nos enviaría en la dirección de la marca externa, pues en ese caso, desde el Q-nodo padre (visitado como paso previo durante la búsqueda) se nos hubiera conducido hasta el Q-nodo que contiene el subárbol extraido.

Con objeto de facilitar la descripción de las acciones necesarias para llevar a cabo la división y traslado de nodos locales al Q-nodo padre, utilizaremos una nomenclatura específica para referirnos a los distintos componentes que intervienen en el proceso. Los términos que utilizamos son los siguientes:

Algoritmo 3-4. Selección del subárbol extraido en Q-nodos del conjunto de regiones

Entrada: Región sobrecargada.

1. De acuerdo a como se describe en el teorema 3-1 y siendo A el árbol k-d local a la región sobrecargada, localizar el primer subárbol S de A que sin satisfacer el requisito de carga, alguno de los hijos la satisface. Sean S_1 y S_2 los subárboles descendientes de S.
2. Si S_1 y S_2 satisfacen el requisito de carga, extraer aquel con menor FCD.
3. Si no extraer aquél que satisfaga el requisito de carga.
4. **Fin**.

- **S-NODO**. El Q-nodo sobrecargado.
- **X-NODO**. El Q-nodo que contendrá el subárbol extraido. Representa la región extraida.
- **MS-NODO**. El Q-nodo resultante después de haber realizado la extracción de un subárbol (el S-NODO Modificado). Este Q-nodo ocupa típicamente la misma dirección física que el S-NODO.
- **P-NODO**. Q-nodo padre del S-NODO que almacena en su árbol local la referencia al S-NODO. Bajo ciertas circunstancias el P-NODO no es único, sino que está compuesto por varios Q-nodos del nivel inmediatamente superior al nivel del S-NODO.
- **s-árbol**. El árbol local del S-NODO. El nodo raíz del s-árbol se denomina *s-raíz*.
- **x-árbol**. El árbol local del X-NODO. La raíz de este árbol es la *x-raíz*.
- **ms-árbol**. Es el antiguo s-árbol después de haberle extraido el x-árbol y haber sido convenientemente modificado. Es el árbol local al MS-NODO.
- **p-árbol**. Es el árbol local al P-NODO.
- **pS-hojas**. Cada una de las referencias al S-NODO que existen en el P-NODO.
- **pX-hojas**. Conjunto de pS-hojas relativas al subárbol extraido. A partir de la pX-hoja serán insertados los nodos trasladados desde el S-NODO hasta el P-NODO.

Aunque existe la posibilidad de que el S-NODO aparezca referenciado desde varios Q-nodos del nivel superior, es decir, que el S-NODO sea un Q-nodo multipadre, con objeto de simplificar nuestra descripción, en las explicaciones que siguen consideramos que el S-NODO tiene un único padre. El caso multipadre será tratado con posterioridad.

Inicialmente y con el ánimo de ilustrar paso a paso el proceso de traslado de nodos locales desde una región sobrecargada hasta el Q-nodo que la gobierna, conviene comenzar por el caso más simple de todos los posibles. Este caso conlleva dos suposiciones iniciales:

- El S-NODO correspondiente a la región sobrecargada no es multipadre. Así el P-NODO es único.
- El p-árbol contiene una única referencia al S-NODO, esto es, no existen referencias redundantes en el P-NODO ,y por tanto, la pS-hoja es única y coincide con la pX-hoja.

Comenzamos la descripción de este primer caso con el apoyo de la figura 3-8, la cual muestra una región R_1 sobrecargada (el S-NODO) y un espacio E que la gobierna. Como puede apreciarse, la situación que muestra la figura satisface las suposiciones anteriores. Con estas consideraciones podemos aplicar los puntos 1 a 4 del proceso de división antes especificados a la región R_1:

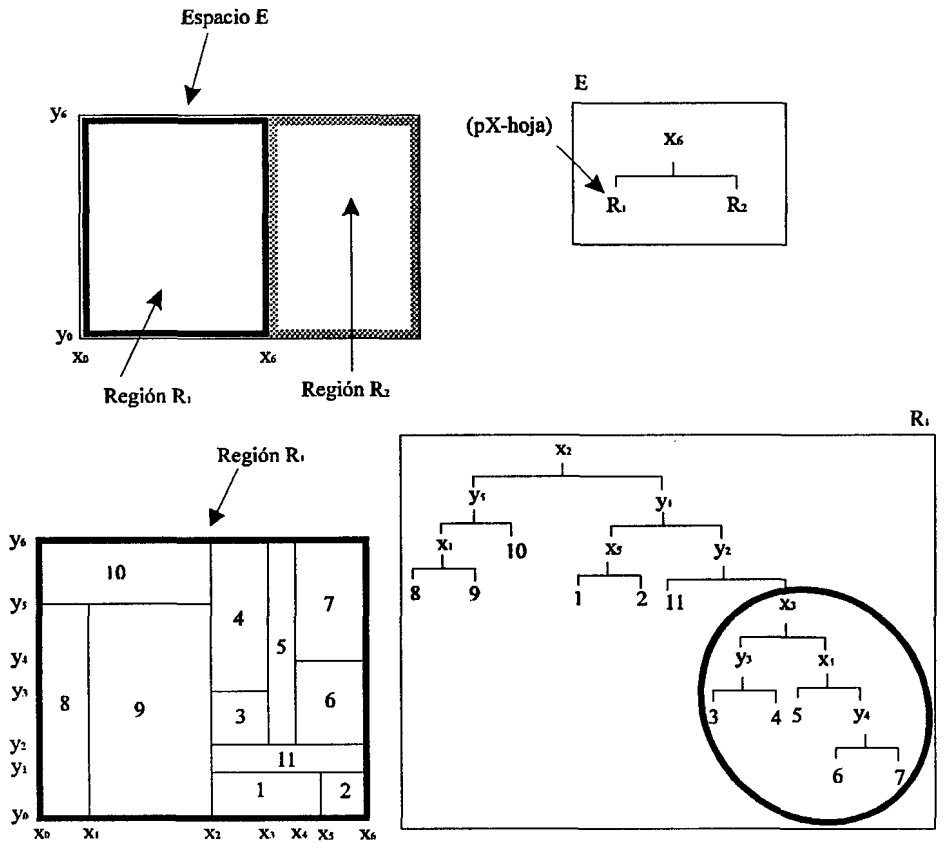


Figura 3-8. Arbol Q con la región R_1 sobrecargada. El espacio E gobierna la región sobrecargada.

1. *Seleccionar el subárbol candidato para la extracción* según el algoritmo 3-4. Este subárbol aparece convenientemente remarcado en el árbol local a R_1 dentro de la figura.

2. *Adjuntar una nueva región* R_3 cuyo árbol local es el subárbol seleccionado en el punto anterior.

3. *Trasladar al P-NODO (E) la frontera* que delimita la región extraida. Esta información viene definida por el camino (x_2 *derecha*, y_2 *derecha*), que es justo el que nos conduce al árbol extraido. Este camino debe ser trasladado hacia el nivel superior del árbol Q, pero antes es preciso llevar a cabo las siguientes acciones:

   3.a *Localizar el Q-nodo padre* que gobierna la región sobrecargada R_1. Teniendo en cuenta el caso tan simple que estamos considerando, en la búsqueda previa a la inserción de la nueva tupla para localizar su contenedor destino, podríamos registrar el espacio de nivel 1 por donde hemos de pasar antes de descender a la región R_1. Este espacio será el P-NODO. Sin embargo, en casos más complejos (por ejemplo en caso de tener un región multipadre) el Q-nodo

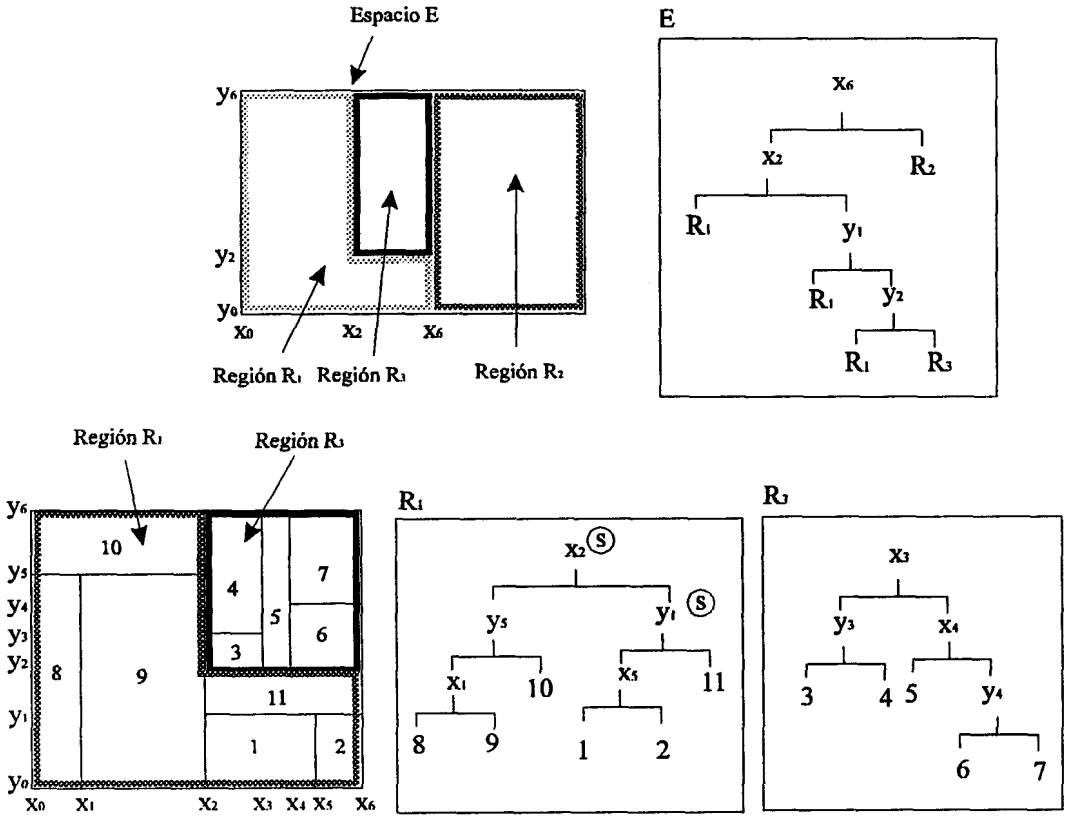


Figura 3-9. Arbol Q resultante después de la división de la región R_1 anteriormente sobrecargada

destino de la información frontera puede no ser único, de modo que este método de localización no sería correcto. Llegado el momento describiremos la forma correcta de llevar a cabo esta localización.

3.b *Identificar el punto dentro del P-NODO (pX-hoja)* a partir del cual hemos de trasladar la información frontera. En este caso, ya que el S-NODO R_1 está referenciado una sola vez dentro de E, localizar este punto es sencillo. El mecanismo descrito en el punto anterior sería válido para identificar la pX-hoja dentro de E. La pX-hoja se encuentra marcada en la figura 3-8.

Una vez identificado el punto dentro de E a partir del cual se inserta la información frontera, el árbol compuesto por el camino (x_2 *derecha*, y_2 *derecha*) se sitúa en la dirección dentro de E que ocupaba la pX-hoja. Lógicamente, como descendientes izquierdo de los dos nuevos nodos locales se coloca una referencia a R_1. Por su parte, el hijo derecho del nodo y_2 será obviamente un apuntador al nuevo X-NODO R_3.

4. *Modificar el s-árbol* para que constituya el nuevo ms-árbol. Esta modificación supone eliminar el nodo padre de la x-raíz (y_2) y su subárbol descendiente por la derecha (justo el x-árbol), colocando en su lugar el subárbol hermano del x-árbol (aquel con raíz x_5).

La figura 3-9 muestra el árbol Q de la figura 3-8 despúes de la división de la región R_1.

Ahora que hemos establecido los puntos básicos en el proceso de división de regiones, podemos ya ampliar nuestro objetivo con el fin de reconocer, a través de situaciones algo más complejas, las implicaciones ulteriores del proceso que acabamos de describir.

Como podemos observar en la figura 3-9, ahora el espacio E que gobierna la región recién dividida R_1, contiene referencias duplicadas a esta región. Si con posterioridad esta región R_1, tuviera que ser de nuevo dividida, es seguro que la raíz del árbol local a R_1, x_2, formaría parte del camino que representa la frontera de la nueva división. Pero este nodo ya ha sido trasladado en la división anterior a E y por consiguiente sería inútil volver a trasladarlo en posteriores divisiones de R_1. Esta consideración nos lleva a marcar de algún modo aquellos nodos locales en el ms-árbol que ya han sido trasladados hacia niveles superiores dentro del árbol Q. Estos nodos locales en el ms-árbol se denominan nodos *subidos*. La definición 3-2 formaliza la consideración de tales nodos.

Los nodos locales subidos no sólo nos permiten evitar la aparición de ciertas redundancias en los Q-nodos de niveles superiores, sino que poseen una serie de propiedades que nos serán de gran utilidad en la descripción y aplicación de los algoritmos de división y traslado de

Definición 3-2. Nodo local subido

Un nodo local a un Q-nodo se dice *subido*, si este nodo ha sido previamente trasladado a algún Q-nodo de nivel superior como resultado de una división. Un nodo local subido incorpora una marca de "subido" para su reconocimiento.

Lema 3-1. Sobre la raíz del ms-árbol

Sea E un Q-nodo no contenedor que se encuentra sobrecargado. La raíz del árbol local a E permanece como raíz de E tras la división si y sólo si la raíz del subárbol extraido no es descendiente directo de la raíz de E.

Demostración.-

s-raiz = ms-raíz ⇒ x-raíz no es descendiente directo de la s-raíz

Dado que el algoritmo de división construye el ms-árbol a partir del s-árbol extrayendo de éste -entre otras modificaciones- el padre de la x-raíz, si s-raíz = ms-raíz la s-raíz no puede ser el padre de la x-raíz y por tanto éste no es descendiente directo de la s-raíz.

x-raíz no es descendiente directo de la s-raíz ⇒ s-raiz = ms-raíz

Si la x-raíz no es descendiente directo de la s-raíz, el padre de la x-raíz no es la s-raíz. Los únicos nodos locales al s-árbol que no aparecen en el ms-árbol son el padre de la x-raíz y los nodos perteneciente al x-árbol. Por tanto, la s-raíz continúa apareciendo como ms-raíz en el ms-árbol. ♦

Proposición 3-1. Marca de subido de la raíz de un Q-nodo

La raíz del árbol local a un Q-nodo Q posee marca de subido si y solo si Q es un nodo multireferenciado, bien desde un solo Q-nodo padre o bien desde varios Q-nodos padre de Q (Q es multipadre).

Demostración.-

raíz de Q subida $\Rightarrow$ *Q multi-referenciado*

Sea Q un Q-nodo y llamemos q-raíz a la raíz de Q. Supongamos que la raíz de Q tiene marca de subido y Q no está multi-referenciado en el nivel superior. En este caso debe existir un único Q-nodo padre conteniendo una única referencia a Q. Además algún Q-nodo E ascendiente de Q contiene la q-raiz como nodo local. Llamemos P al Q-nodo que contiene la referencia Q y R al Q-nodo que contiene la q-raíz subida. La referencia Q dentro de P es necesariamente descendiente de la q-raíz en R. Supongamos que nos situamos en la q-raíz dentro de R. Para llegar a la referencia Q de P debemos seguir un único camino, ya que por hipótesis, Q no está multi-referenciado. Si es el izquierdo, esto significa que la zona del espacio que abarca Q se localiza a un lado del hiperplano que define la q-raiz. Si el camino a seguir es el derecho, la región que abarca Q se encuentra al otro lado del hiperplano definido por la q-raíz. Pero esto es una contradicción ya que la raíz en Q tendrá al menos dos hijos, indicando que Q abarca zonas a un lado y otro del hiperplano definido por la q-raíz.

Q multi-referenciado $\Rightarrow$ *raíz de Q subida*

Supongamos ahora que la raíz de Q no tiene marca de subido y veamos entonces que el Q no puede estar multi-referenciado. Si la q-raíz no tiene marca de subido es porque Q se encuentra en alguno de estos dos casos:

1. Q nunca ha sido dividido, o bien
2. Durante el proceso de división, la q-raíz desaparece de Q. Pero según el lema anterior esto equivale a que Q ha sido previamente dividido extrayendo algún subárbol descendiente directo de la s-raíz.

Estudiemos ambos casos por separado.

1 Q nunca ha sido dividido. Ahora bien, un Q-nodo que nunca antes ha sido dividido atiende a dos circunstancias excluyentes:

1.a Representa a todo el dominio de búsqueda, o bien

1.b Representa una región extraida del dominio de la cual nada se ha extraido con posterioridad.

Analicemos ambas circunstancias.

1.a Si representa a todo el dominio, entonces Q no tiene padre y por tanto no puede estar multi-referenciado.

1.b Si representa una región extraida del dominio de búsqueda, Q no puede estar multi-referenciado, pues el algoritmo de división coloca en el Q-nodo padre una única referencia a la región extraida. Dado que Q no volvió a ser un S-NODO ninguna referencia adicional a él pudo ser trasladada al padre.

2. Q fue en algún momento previo un S-NODO del cual se extrajo un subárbol descendiente directo de la antigua s-raíz, es decir la antigua raíz de Q.

Si Q sólo ha sufrido una división previa, por estar multi-referenciado, más de un nodo fue trasladado, pero esto entra en contradicción con la hipótesis de que el árbol extraido era descendiente directo de la s-raíz.

$\Rightarrow$

...

Supongamos entonces que tenemos un Q-nodo Q multi-referenciado que ha sufrido dos divisiones ($Q_0 \rightarrow Q_1 \rightarrow Q$) y está de nuevo sobrecargado.Tras la primera división Q_1 no quedó multi-referenciado según lo anterior. Así pues fue a partir de la segunda división ($Q_1 \rightarrow Q$) cuando Q quedó multi-referenciado. Pero para que esto sea posible, según el algoritmo de división es necesario trasladar al Q-nodo padre de Q al menos dos nodos locales a Q_1. Sin embargo, según la hipótesis, en la última división ($Q_1 \rightarrow Q$) a Q_1 se le extrajo un subárbol descendiente directo de la raíz, lo cual implicaría que sólo el nodo raíz tendría que haber sido trasladado al Q-nodo padre de Q, con lo que llegamos a una contradicción.♦

nodos locales, tanto a nivel de regiones como a nivel de espacios. Las proposiciones 3-1 y 3-2 revelan propiedades muy interesantes acerca de los nodos subidos. El lema 3-1, por su parte, enuncia un resultado previo de aplicación en la demostración de la proposición 3-1.

Los resultados referentes a las proposiciones 3-1 y 3-2 nos proporcionan una información muy valiosa en dos aspectos fundamentales del proceso de división de un Q-nodo.

- Si la s-raíz es un nodo local subido, en virtud de la proposición 3-1, existen varias referencias al S-NODO en uno o más Q-nodos del nivel superior. Este apunte hace necesario revisar los puntos 3.a y 3.b en el proceso de división de regiones ya que, en este caso, el P-NODO puede no ser único y las pS-hojas son múltiples. En caso de que la s-raíz sea no subido, estamos seguros de que la pS-hoja es única y se encuentra por tanto en un único P-NODO. Los puntos 3.a y 3.b pueden aplicarse tal y como se describieron previamente.
- Dado que en el proceso de traslado de nodos locales hacia el P-NODO, todos los nodos comprendidos entre la s-raíz y el padre de la x-raíz son trasladados hacia arriba, es claro que todos los ascendientes de un nodo subido son necesariamente nodos subidos. Si el padre de la x-raíz es un nodo local subido, no será preciso entonces trasladar ninguna información frontera hacia el P-NODO (lo cual no nos libera de efectuar determinados cambios en las pX-hojas). Así pues, sólo en el caso de que el padre de la x-raíz sea un nodo local no subido, será necesario trasladar nodos hacia el nivel superior. Pero, de acuerdo con la proposición 3-2, en estas circunstancias la pX-hoja es única, lo que simplifica el algoritmo de traslado de nodos.

Teniendo en cuenta la importancia de reconocer nodos locales subidos, cada nodo local debe incorporar una marca que, como paso final del algoritmo de división de Q-nodos, se activa para aquellos nodos en el ms-árbol que hayan sido trasladados al P-NODO. En la figura 3-9, los nodos subidos pertenecientes al MS-NODO R_1 están marcados con el símbolo *s* englobado en un pequeño círculo.

Las propiedades de los nodos locales subidos y sus implicaciones nos llevan a reflexionar de nuevo acerca de la mejor elección para el subárbol extraido. Si el subárbol candidato para ser extraido tiene como raíz un nodo local cuyo padre es subido, no es necesario trasladar al

Proposición 3-2. Propiedades de la x-raíz

Si la x-raíz es un nodo no subido, la pX-hoja es única. Por contra, si la x-raíz es un nodo subido, existen varias pX-hojas en el árbol Q.

Demostración.

Supongamos que la x-raíz es un nodo no subido. El algoritmo de división extrae el x-árbol y lo coloca como árbol local de un nuevo Q-nodo. Este nuevo Q-nodo posee una raíz no subida y según la proposición 3-1, este Q-nodo no puede estar multi-referenciado. Por su parte, si este nuevo Q-nodo que contiene el x-árbol no está multi-referenciado, significa que desde la raíz del árbol Q sólo hay un camino posible para llegar a este Q-nodo y, en consecuencia, a la zona que este Q-nodo gobierna. Por tanto, si tomamos un punto interior a la zona que gobierna el x-árbol y realizamos una búsqueda por el árbol Q, llegamos necesariamente a una única pS-hoja, que es la pX-hoja.

El razonamiento contrario se aplica para demostrar la segunda parte de la proposición. ♦

Q-nodo padre ninguna información frontera adicional. Esta circunstancia, además de simplificar el algoritmo de división, supone la ventaja de no incrementar el volumen de datos en la zona de índices del árbol Q.

En realidad, la elección de un subárbol cuyo nodo local padre sea subido significa que las fronteras previamente establecidas para separar regiones (espacios si nos encontramos en niveles superiores del árbol Q), siguen siendo válidas para discernir el contenido de nuevas regiones. Esta situación es típica en estructuras multidimensionales tales como los ficheros rejilla [52], donde cada división provoca un particionamiento en cascada de todo el dominio de búsqueda . En los casos en que es necesaria una nueva división, ésta se intenta por fronteras antes establecidas con objeto de no provocar nuevas particiones en cascada.

De acuerdo a la discusión previa es preciso modificar levemente el algoritmo 3-4 de selección de subárbol extraido para tomar en consideración el nuevo objetivo planteado, a saber, que la x-raíz sea un nodo local subido. El nuevo algoritmo de selección actúa según el planteamiento genérico que se muestra bajo el epígrafe del algoritmo 3-5.

Con las últimas definiciones y valoraciones incorporadas tras el primer ejemplo de división de una región, estamos ya en condiciones de describir con un mayor grado de realismo el proceso de división de regiones. Antes de facilitar el algoritmo de división y traslado de nodos locales, ilustraremos el proceso con un ejemplo de división de una región en un escenario algo más complejo que el caso previamente ilustrado.

Consideremos el árbol Q de la figura 3-10, en la que se muestra el espacio E y la región R_1 gobernada por E. Supongamos que E es la raíz del árbol Q. Supongamos además que R_1 es un S-NODO y debe ser por tanto dividido. Notemos que la raíz x_2 del s-árbol es un nodo local subido y, consecuentemente, R_1 aparece referenciado más de una vez dentro de E.

Algoritmo 3-5. Selección de subárbol extraido para regiones

Entrada: Región sobrecargada.

1. $\mathscr{C}$= {Subárboles del s-árbol que cumplen el requisito de carga}. $\mathscr{C}\neq\emptyset$
2. $\mathscr{S}$= { S ∈ $\mathscr{C}$ / S tiene como padre un nodo local subido}
3. Si $\mathscr{S}\neq\emptyset$ (No conlleva traslado de nodos al P_NODO)
4. Seleccionar S ∈ $\mathscr{S}$ / FCD(S) = Min (FCD(S_i)), ∀ S_i ∈ $\mathscr{S}$
5. Si $\mathscr{S}$= ∅ (Conlleva traslado de nodos locales al P_NODO)
6. $\mathscr{AM}$ = {S ∈ C / Altura(Raíz de S) = Min(Altura raíz (S_i)), ∀ S_i ∈ $\mathscr{C}$}, (1≤Card($\mathscr{AM}$)≤2)

Analicemos dos casos diferentes en función del árbol que se selecciona para ser extraido de R_1 y estudiemos las consecuencias que cada caso acarrea. Primero analizamos el caso de un árbol extraido cuya raíz es un nodo local no subido y, en segundo lugar, evaluamos las consecuencias de extraer un subárbol cuya raíz es un nodo local subido.

Caso 1. Extracción del subárbol B de R_1. Como la x-raíz no es un nodo subido, a pesar de que las pS-hojas son múltiples (la s-raíz es un nodo subido), en virtud de la proposición 3-2, la pX-hoja es única. A diferencia de como arguíamos en el ejemplo anterior y debido a que la s-raíz es un nodo subido, esta pX-hoja puede no corresponderse con la pS-hoja visitada en el recorrido previo para localizar el contenedor destino de la tupla que provocó la división del contenedor y la posterior sobrecarga de la región R_1. Basta considerar el caso de una tupla que provocó la división de un contenedor localizado en la zona que gobierna el árbol A dentro de R_1. El camino previo para localizar dicho contenedor pasa por la pS-hoja situada como descendiente izquierdo del nodo x_2 local al p-árbol (árbol k-d local a E). Dado que el subárbol a extraer es B, la pX-hoja es aquella que nos conduce a la zona que gobierna dicho subárbol,

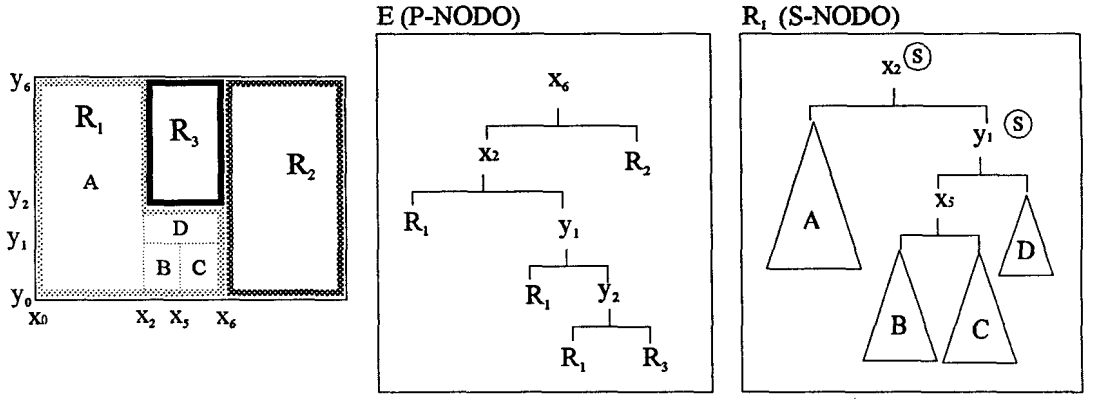


Figura 3-10. Ejemplo de árbol Q original. Se muestran la región R_1 en situación de sobrecarga, y el espacio E que la gobierna.

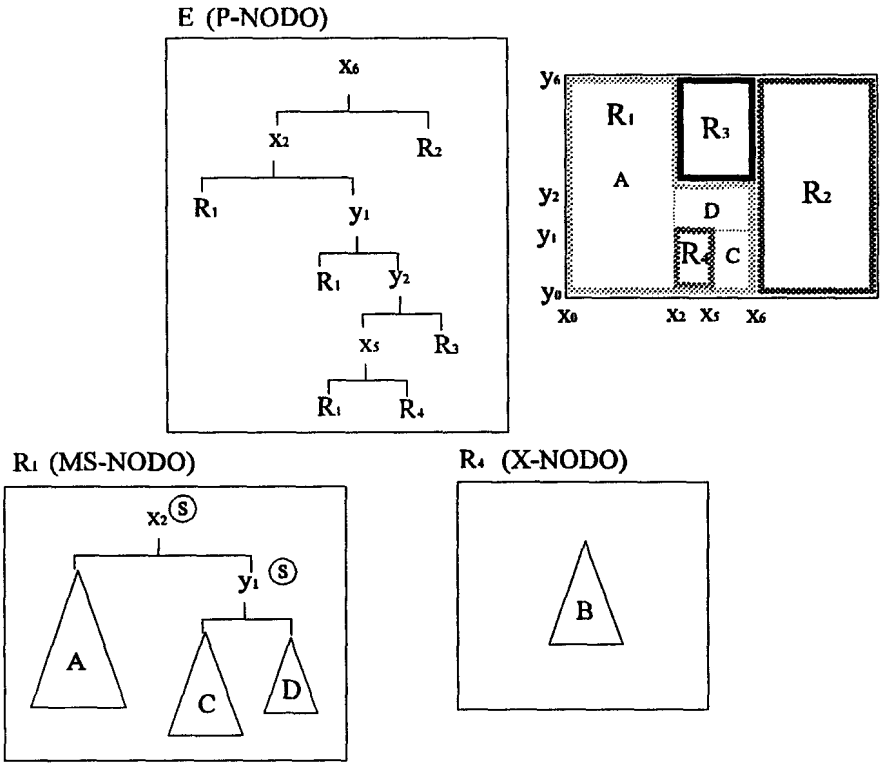


Figura 3-11. Arbol Q resultante tras la extracción del subárbol B de R_1.

esto es el descendiente izquierdo del nodo y_1 local a E.

Es preciso entonces habilitar un mecanismo que nos permita identificar la pX-hoja cuando ésta es única. El método para localizar la pX-hoja implica realizar un nuevo recorrido por el árbol Q, distinto al que se lleva a cabo para insertar la tupla que provoca la división. Este recorrido comienza en una hoja arbitraria interior al x-árbol. Esta hoja apuntará a un contenedor de donde una tupla arbitraria debe ser obtenida. Con esta tupla y comenzando por el Q-nodo raíz, realizamos una búsqueda exacta de la misma hasta llegar a una hoja en el P-NODO conteniendo la dirección del S-NODO. Esta es la pX-hoja.

Una vez localizada la pX-hoja, la información frontera debe ser trasladada a partir de la pX-hoja. Como ya sabemos, esta información frontera consiste en aquellos nodos locales no subidos en el camino desde la s-raíz hasta el padre de la x-raíz. Para efectuar este traslado, nos ayudamos de dos cursores, uno para el p-árbol (el p-cursor) y otro para el s-árbol (el s-cursor). El p-cursor se inicializa para que apunte a la pX-hoja. El s-cursor se sitúa sobre la s-raíz. El nodo actual sobre el que se sitúa el p-cursor se denomina p-nodo mientras que el nodo sobre el que se sitúa el s-cursor se denomina s-nodo. La dirección que conduce al x-árbol se denomina x-dirección, la dirección opuesta se denomina o-dirección.

El proceso de traslado actúa como sigue. Si el s-nodo es un nodo subido, avanzamos el s-cursor en la x-dirección. En caso contrario, copiamos el s-nodo sobre la dirección apuntada por el p-cursor. Como descendiente del p-nodo en la o-dirección colocamos una referencia al S-NODO. Finalmente avanzamos ambos cursores en la x-dirección. Este proceso se repite hasta que el s-cursor apunte a la x-raíz. En este momento, en la dirección apuntada por el p-cursor se sitúa una referencia al nuevo X-NODO.

El punto 4 del proceso de división de una región es invariante, por lo que puede ser aplicado sin modificación alguna. El árbol Q resultante después de esta división se muestra en la figura 3-11.

Caso 2. Extracción del subárbol con raíz y_1 en R_1. En este caso, como muestra la figura 3-10, la x-raíz es un nodo subido. Esta circunstancia significa, desde un punto de vista geométrico, que las zonas del dominio de búsqueda que gobierna el x-árbol se encuentran separadas por fronteras establecidas en divisiones previas. Desde un punto de vista estructural, esto significa que las pX-hojas son múltiples, esto es, para alcanzar toda la zona gobernada por el x-árbol, es preciso derivar por distintas hojas en el p-árbol.

La localización en este caso de las pX-hojas, consiste en obtener tuplas de las distintas zonas del dominio de búsqueda gobernado por el x-árbol que se encuentren a uno y otro lado de alguna frontera previa. La obtención de estas tuplas se lleva a cabo como sigue.

Comenzando en la x-raíz, bajamos por ambos hijos. Cuando alcancemos un nodo local no subido, bajamos hasta una hoja arbitraria descendiente de este nodo. Esta hoja albergará un apuntador a un contenedor. Si, por el contrario nos encontramos ante un nodo local subido descendemos de nuevo por ambos hijos, hasta alcanzar nodos locales no subidos o bien hojas del x-árbol. Para cada hoja alcanzada en este recorrido, accedemos al contenedor que referencia y tomamos una tupla arbitraria. Con cada una de estas tuplas realizamos ahora un recorrido por el árbol Q, comenzando por la raíz del mismo. Este recorrido puede hacerse conjunto para todas las tuplas obtenidas justo hasta alcanzar el P-NODO (recordemos que actualmente consideramos que la región sobrecargada es un Q-nodo no multipadre), dentro del cual llegará un punto en el que se bifurque hacia tantas hojas como tuplas busquemos. Todas estas hojas son registradas como pX-hojas.

En nuestro ejemplo, las pX-hojas se localizan como hijo izquierdo de los nodos locales y_1 e y_2 dentro de E.

Dado que no es preciso trasladar ningún nodo local desde el s-árbol hacia el p-árbol (todos los nodos en el camino han sido anteriormente subidos), la única modificación que debe tener lugar dentro del p-árbol consiste en modificar las pX-hojas que referencian al S-NODO R_1 para que referencien tras la división al X-NODO R_4. En la figura 3-12 se muestra el resultado el árbol Q resultante tras la división.

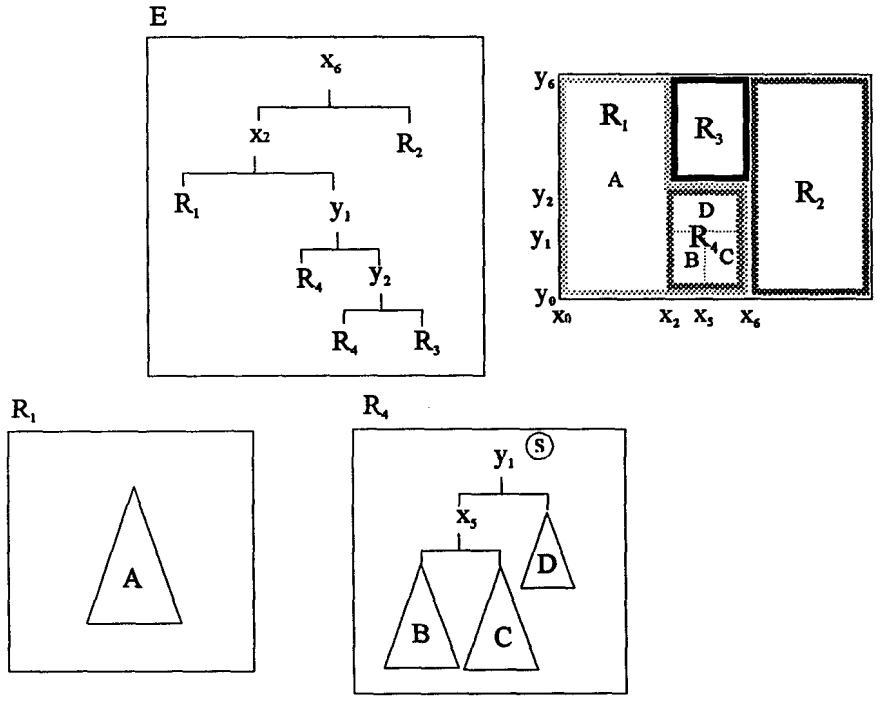


Figura 3-12. Arbol Q resultante tras la extracción del subárbol con raíz x_5 en R_1.

Si prestamos atención al significado de que la x-raíz sea un nodo subido, podemos deducir que mientras que el S-NODO sea no multipadre, una copia de la x-raíz debe encontrarse necesariamente en el P-NODO. Aunque con posterioridad se establecerá formalmente, resulta evidente que las pX-hojas de un tal S-NODO deben ser descendientes de esta copia de la x-raíz en el p-árbol. Esto significa que, identificando de forma única a los nodos locales de un región, podríamos evitar el tener que extraer tuplas a partir de ciertos contenedores para luego descender por el árbol Q en busca de estas tuplas hasta localizar las pX-hojas. En su lugar, una sola tupla arbitraria gobernada por el x-árbol sería suficiente para realizar el recorrido por el árbol Q hasta alcanzar la copia de la x-raíz. A partir de dicho nodo local en el p-árbol, todas las pS-hojas descendientes son pX-hojas. Pero este método será convenientemente descrito una vez que hayamos estudiado con detenimiento la división de espacios.

Después de haber ilustrado los diferentes casos que pueden presentarse en la división de regiones, por una parte, podemos presentar formalmente un algoritmo genérico de identificación de pX-hojas en el caso de regiones y, por otra, aportar el algoritmo de traslado de nodos locales desde el s-árbol hasta el p-árbol. Los algoritmos 3-6 y 3-7 describen, respectivamente, ambos procesos.

Algoritmo 3-6. Identificación genérica de pX-hojas en regiones

Entrada: s-árbol.

1. Llamemos *n* al nodo local en el que estamos actualmente situados. Inicialmente *n* = x-raiz dentro del s-árbol.
2. **si** *n* es una hoja
3. acceder al contenedor apuntado por *n* y extraer de él una tupla arbitraria. Ir a 5.
4. **si no si** *n* es un nodo local subido
5. bajar a ambos hijos y repetir el paso 2 para cada nodo hijo
6. **si no**
7. bajar al ramal de menor altura y hacer *n* = nodo local alcanzado. Ir a 2.
8. **fin si**
9. **para** cada tupla obtenida hacer
10. recorrer el árbol Q hasta localizar todas las referencias al S-NODO. Registrar estas referencias como pX-hojas.
11. **fin para**
12. **Fin.**

Algoritmo 3-7. Traslado de nodos locales hacia el p-árbol

Entrada: s-árbol y p-árbol.

1. Se utilizan dos cursores para recorrer el s-árbol y el p-árbol. El s-cursor llevará la pista de los nodos visitados en el s-árbol y el p-cursor nos ayudará a recorrer el p-árbol. El p-cursor se inicializa para que apunte a la(s) pX-hoja(s). El s-cursor se sitúa sobre la s-raíz. El nodo actual sobre el que se sitúa el p-cursor se denomina p-nodo mientras que el nodo sobre el que se sitúa el s-cursor se denomina s-nodo. La dirección que conduce al x-árbol se denomina x-dirección, la dirección opuesta se denomina o-dirección.
2. **si** la pX-hoja no es única
3. modificar cada pX-hoja por un apuntador al X-NODO
4. **return**
5. **si** el s-cursor no apunta a la x-raíz
6. **si** el s-nodo es subido
7. avanzar el s-cursor en la x-dirección.
8. **si no**
9. colocar una copia del s-nodo sobre la dirección apuntada por el p-cursor
10. marcar el s-nodo dentro del s-árbol como nodo local subido
11. colocar como hijo del p-nodo en la o-dirección un apuntador al S-NODO
12. avanzar ambos cursores en la x-dirección
13. **fin si**
14. **si no**
15. colocar en la dirección apuntada por el p-cursor un apuntador al X-NODO
16. **fin si**
17. **Fin.**

Recordemos antes de finalizar este apartado que, durante el proceso ilustrado de división de regiones, hemos considerado siempre que la región sobrecargada es un Q-nodo no multipadre. Esto significa que todas las pX-hojas implicadas en el proceso de traslado de nodos locales se localizan en un único P-NODO. Veamos para concluir este apartado qué sucede en el caso de que el S-NODO sea un Q-nodo multipadre.

Consideremos de nuevo la figura 3-10. Supongamos ahora que el Q-nodo del conjunto de espacios E se divide, de modo que el árbol seleccionado para ser extraido es áquel enraizado en y_2. Fijémonos que las referencias a R_1 englobadas en el árbol que va a ser extraido de E nos conducen a zonas diferentes gobernadas por R_1. Por una parte, la referencia a R_1 situada como hijo izquierdo de y_1 nos conduce a la zona de R_1 gobernada por los subárboles B y C. Por su parte la referencia situada como hijo de y_2 en E nos conduce a la zona de R_1 gobernada por el subárbol D. Esta división de E provocaría que la región R_1 se convirtiera en un Q-nodo multipadre.

Si con posterioridad R_1 fuera dividido extrayendo un subárbol que englobase a los subárboles B, C y D, seleccionando por ejemplo como subárbol extraido áquel que tiene como raíz a y_1, las pX-hojas se localizarían en dos P-NODOS diferentes. Los algoritmos 3-4 y 3-5 de identificación de pX-hojas y traslado de nodos al p-árbol, respectivamente, son perfectamente válidos también para este caso. La diferencia estriba en el coste de dichos algoritmos, ya que en este caso dos o más Q-nodos en el nivel superior son implicados en el proceso de división. Lógicamente y según se estableció en la proposición 3-2, si la raíz del arbol seleccionado para ser extraido no es un nodo subido, el hecho de que el S-NODO sea multipadre no afecta para nada al proceso de división establecido previamente, ya que un solo P-NODO está implicado en el proceso de división.

Como conclusión de todo lo anterior podemos afirmar que el proceso de división de regiones y posterior traslado o modificación de nodos locales en el P-NODO no resulta muy costoso mientras seamos capaces de mantener las regiones del árbol Q como Q-nodos no multipadre. Con este objetivo en mente, el algoritmo final de división de regiones debe incluir algún método para marcar los nodos locales del Q-nodo que gobierna a la región dividida, de modo que nos permita evitar que una posterior división de éste Q-nodo provoque la condición de multipadre para alguna región del árbol Q. Pero la forma en la que deben ser marcados los nodos locales del P-NODO se describirá en el siguiente apartado, donde estudiamos la división de espacios.

4.3 División de espacios

Con la información que poseemos sobre el significado de los Q-nodos a distintos niveles en el árbol Q, podemos decir sin temor a equívocos que la diferencia esencial, desde un punto de vista estructural, entre un Q-nodo del conjunto de regiones y un Q-nodo del conjunto de espacios, radica en la existencia dentro de estos últimos de referencias redundantes a Q-nodos

descendientes. Como hemos visto en el apartado anterior, la existencia de esta redundancia puede en un futuro provocar la aparición de Q-nodos multipadre del nivel inferior y encarecer con ello los algoritmos de división.

El algoritmo 3-5 de selección de subárboles extraidos para regiones incorpora ya medidas para reducir en lo posible las duplicaciones que pueden aparecer en espacios de nivel superior. Sin embargo estas medidas deben ser reforzadas con objeto de evitar la creación de Q-nodos multipadre.

Es conveniente apuntar que, por una parte, tal y como hemos visto en el apartado anterior, la aparición de Q-nodos multipadre encarece el coste de los algoritmos de división. Por otra parte, no menos importante, los Q-nodos multipadre (los cuales han sido fuente de errores en anteriores versiones del árbol Q así como en otras estructuras multidimensionales como el árbol hB) dificultan los algoritmos de borrado en el árbol Q y por tanto su adaptabilidad.

Esta circunstancia hace necesario disponer de alguna técnica que impida extraer (siempre que ello sea posible) del interior de un espacio, subárboles cuyas hojas contengan referencias que se repiten **fuera** del subárbol extraido. Si impedimos que un árbol así pueda ser extraido, evitamos la aparición de nodos multipadre. Así por ejemplo, en la figura 3-10, si E tuviera que ser dividido, sería importante no extraer el subárbol enraizado en y_1 ó y_2, pues ello provocaría que R_1 se convirtiera en un nodo multipadre.

El método propuesto consiste sencillamente en utilizar marcas en los nodos locales a espacios que nos informen de esta circunstancia. Estas marcas se denominan marcas de inclusión.

Resulta claro entonces que la colocación de marcas de inclusión en nodos locales a Q-nodos del conjunto de espacios debe formar parte del proceso de división de cualquier Q-nodo de la zona de índices, ya sean regiones o espacios.

Así pues, considerando las diferencias existentes entre espacios y regiones y con el fin de acercarnos al establecimiento del algoritmo definitivo de división de Q-nodos, las modificaciones, así como las nuevas aportaciones a los métodos previamente descritos involucrados en la división de regiones pueden resumirse en los siguientes puntos:

- Aportación de un algoritmo de colocación de marcas de inclusión en nodos locales al (a los) P-NODO (S).
- Modificación del algoritmo de selección de subárbol extraido con objeto de no provocar la aparición de Q-nodos multipadre.
- Establecimiento de un algoritmo de colocación de marcas de identificación de Q-nodos multipadre.
- Reconversión del algoritmo de identificación de pX-hojas con el fin de simplificar este proceso para el caso de Q-nodos no multipadre.

Lema 3-2. Propiedades del subárbol más pequeño conteniendo las pS-hojas

Sea M un Q-nodo no multipadre que aparece multi-referenciado en el árbol local a su Q-nodo padre P. Entonces, el subárbol más pequeño S dentro del árbol local a P que engloba todas las referencias a M tiene como raíz a una copia de la raiz de M.

Demostración.-

Como por hipótesis M es un nodo multi-referenciado, de acuerdo con la proposición 3-1, el árbol local a M tiene en su raíz una marca de subido.

Cuando la raíz de M fue trasladada hasta P, sólo existía previamente una única referencia a M en P (o ninguna si P no existía), ya que un Q-nodo con raíz no subida no es multi-referenciado. Según el algoritmo 3-7 de traslado de nodos al p-árbol, la raíz de M se coloca en la posición previamente ocupada por la referencia a M en el árbol local a P. Esto significa que fuera del subárbol definido por la raíz trasladada no hay referencias a M. La cuestión es que tampoco puede haberlas en un futuro, ya que todo lo que se traslade desde M hacia el padre debe hacerse a partir de alguna hoja que referencie a M. Además nodos locales trasladados desde otros Q-nodos distintos de M nunca pueden referenciar a M, pues el algoritmo de traslado inserta o bien referencias al S-NODO o bien al X-NODO, y el X-NODO siempre ocupa nuevas direcciones no existentes previamente en el árbol Q.

Según lo visto, fuera del subárbol definido por la raíz de M trasladada a P no hay referencias a M. Por tanto, si una copia de la raíz de M se encuentra en el árbol local a P, esta copia define un subárbol que incluye todas las referencias a M.

Supongamos que la raíz de S no es la raíz de M. Como M está multi-referenciado en P, la raíz de M es un nodo subido y una copia de ésta se encuentra en P. Sea *m* el nodo local que contiene dicha copia. Según hemos visto previamente, el subárbol que define esta copia incluye todas las referencias a M, por tanto S es interior a este subárbol. Como S es el más pequeño subárbol en P albergando todas las referencias a M, uno de los hijos de *m* en P no incluye referencias a M, lo cual significa que la zona del domino de búsqueda que M gobierna se encuentra a uno u otro lado del hiperplano define *m*. Pero esto entra en contradicción con el hecho de que *m* sea raíz de M. ♦

Proposición 3-3. Propiedades de un S-NODO no multipadre cuando la x-raíz es no subido

Sea S un S-NODO no multipadre. Si la x-raíz es un nodo subido, existe en el p-árbol de S un subárbol que engloba a todas las pX-hojas, tal que no contiene otras pS-hojas que las pX-hojas. La raíz de este subárbol es la copia de la x-raíz en el p-árbol.

Demostración.

Si la x-raíz es un nodo subido, el X-NODO llegará a tener como raíz un nodo subido. Esto significa que este X-NODO aparecería multi-referenciado en el p-árbol. Según el lema anterior, el subárbol más pequeño dentro del p-árbol que engloba todas las referencias al X-NODO (es decir todas las pX-hojas) tiene como raíz a una copia de la x-raiz. Llamemos C a este subárbol.

Por otra parte, por ser S no multipadre, y ser la x-raiz un nodo local subido, una copia del padre de la x-raíz se encuentra también en el p-árbol y necesariamente esta copia es ascendiente de la copia en el p-árbol de la x-raíz. Llamemos *p* a este nodo del p-árbol. Supongamos que la x-raíz es hijo izquierdo de este nodo. Esto significa que la región que gobierna el X-NODO se encuentra al lado izquierdo del hiperplano que define *p*.

⇒

...

Supongamos que C incluye referencias al MS-NODO, es decir C incluye otras pS-hojas distintas de las pX-hojas, esto significa que ciertas zonas del espacio gobernadas por el MS-NODO se hallan a la izquierda del hiperplano que define *p*. Pero esto es una contradicción ya que la x-raíz era hijo izquierdo de *p*, es decir, a la izquierda del hiperplano que define *p* está la zona que gobierna el X-NODO y por tanto ha sido extraida completamente del MS-NODO. Similar razonamiento se aplica si la x-raíz era el hijo derecho de *p*.♦

Antes de tratar individualmente cada uno de estos puntos, precisamos de dos resultados preliminares que serán de utilidad en los siguientes apartados. Estos resultados establecen algunas propiedades que satisfacen los Q-nodos no multipadre. El lema 3-2 y la subsiguiente proposición 3-3 enuncian y demuestran propiedades acerca de Q-nodos no multipadre que nos facilitarán el establecimiento de diversos mecanismos en el proceso de traslado de nodos.

A continuación tratamos cada uno de los puntos anteriores en un apartado difierente.

4.3.1 Nodos de inclusión

Antes de describir el método para colocar marcas de inclusión, conviene formalizar el concepto de *nodos de inclusión*.

Definición 3-3. Nodos de inclusión

Sea E un Q-nodo del conjunto de espacios. Sea *n* un nodo del subárbol local a E y sea N el subárbol enraizado en *n*. El nodo local *n* se define como *nodo de inclusión* si ninguna de las hojas de N aparece duplicada fuera de N.

Para ilustrar la forma en que las marcas de inclusión deben ser colocadas en los Q-nodos del conjunto de espacios y con objeto de mostrar el abanico de situaciones que pueden presentarse, vamos a considerar cuatro casos diferentes. En todos estos casos consideramos que el Q-nodo afectado de sobrecarga no es multipadre.

Caso 1. La s-raíz no tiene marca de subido.

Consideremos el árbol Q de la figura 3-13. En virtud de la definición anterior los nodos de inclusión de E aparecen marcados con el símbolo *i* encerrado en un pequeño círculo.

Supongamos que la región R_2 debe ser dividida y que el subárbol G dentro de R_2 es seleccionado como subárbol extraido. Como la x-raíz no es un nodo subido (los antecesores no tienen marca de "subido"), la pX-hoja de R_2 es única. De hecho, dado que la s-raíz no es un nodo subido, sólo existe una referencia a R_2 dentro de E. Esto significa que el primer nodo local que traslademos desde R_2 hasta E será tal que fuera del subárbol que define este nodo

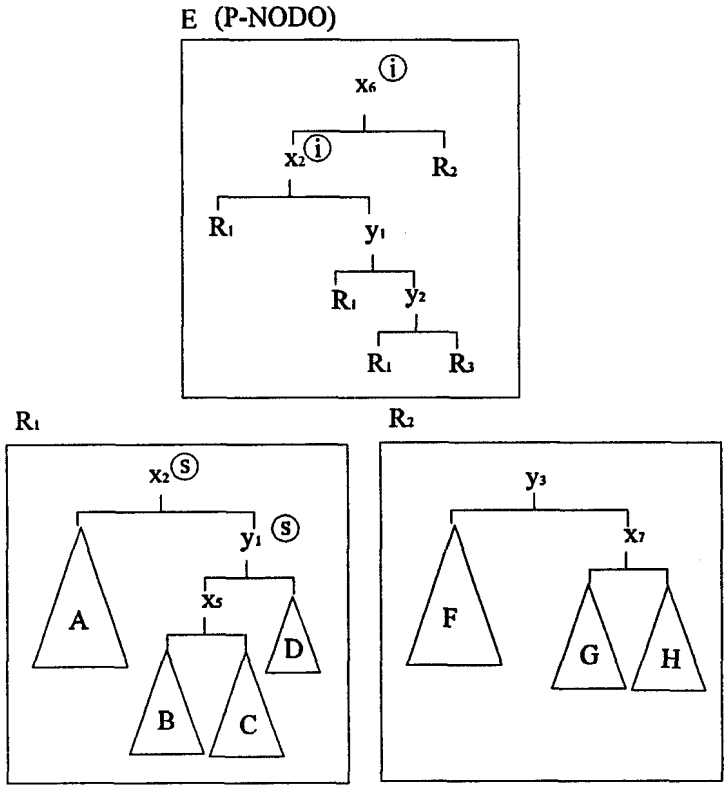


Figura 3-13. Arbol Q en el que se muestra un espacio E con dos nodos locales de inclusión y dos regiones R_1 y R_2 gobernadas por E.

local no pueden existir referencias a R_2. Esta consideración nos llevaría a **marcar el primer nodo que subamos como un nodo de inclusión**. El resultado de la división se muestra en la figura 3-14.

Caso 2 La s-raíz posee marca de subido. El padre de la x-raíz no tiene marca de subido. (Al menos un nodo local debe ser trasladado al P-NODO).

Miremos nuevamente el árbol Q de la figura 3-13. Supongamos ahora que la región sobrecargada es R_1 y que seleccionamos B como subárbol para ser extraido de R_1.

En este caso, dado que la s-raíz es un nodo subido, el P-NODO E incluye múltiples referencias a R_1. Sin embargo la x-raíz no es un nodo subido, por lo que la pX-hoja es única y se localiza como hijo izquierdo del nodo y_1 en E. El algoritmo de traslado de nodos nos llevaría a trasladar el nodo local padre de B, x_5, y colocarlo a partir de la pX-hoja. Notemos que en este caso el nodo trasladado no es un nodo de inclusión y, como tal, **el nodo trasladado no debe ser marcado como nodo de inclusión**. El resultado de la división se muestra en la figura 3-15.

Caso 3. La s-raíz posee marca de subido. La x-raíz no tiene marca de subido pero el padre de la x-raíz es un nodo subido. (Ningún nodo local debe ser trasladado al P-NODO).

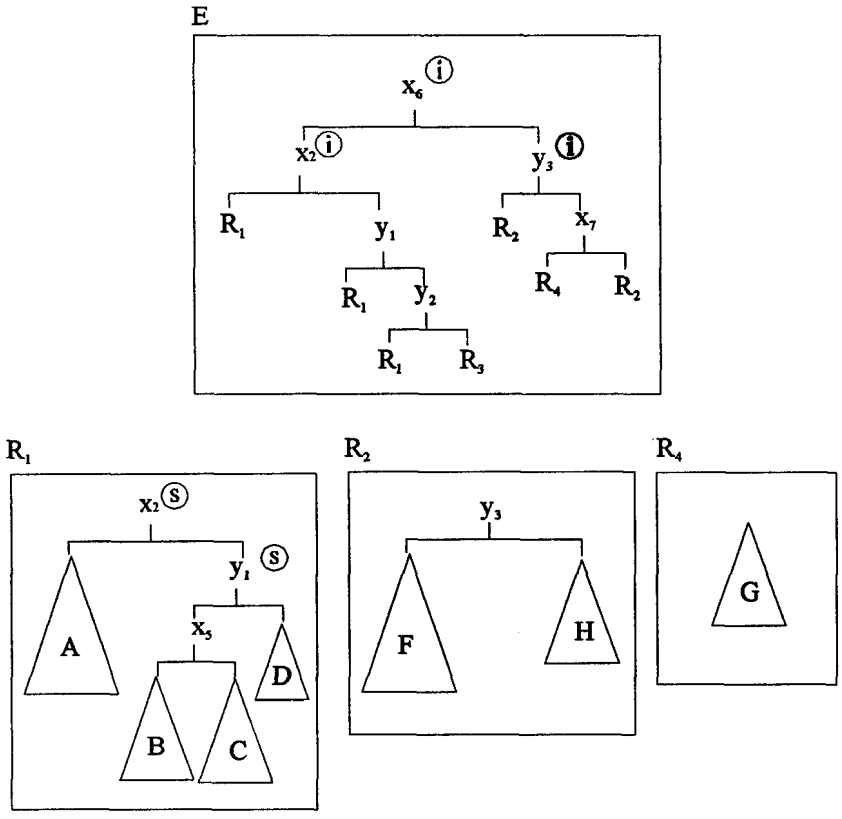


Figura 3-14. Arbol Q resultante producto de la división de R_2 tras la extracción del subárbol G en dicha región.

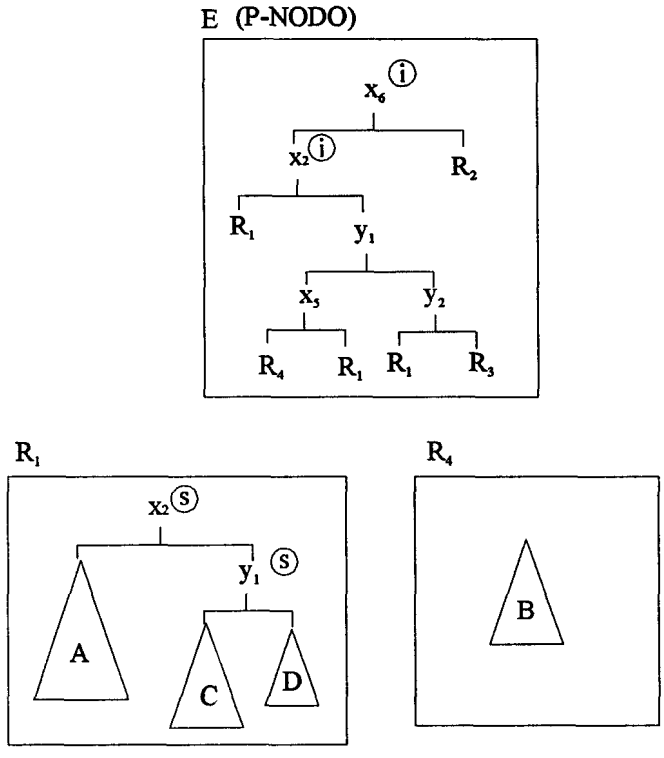


Figura 3-15. Arbol Q resultante tras la división de R_1 tomando B como subárbol extraido.

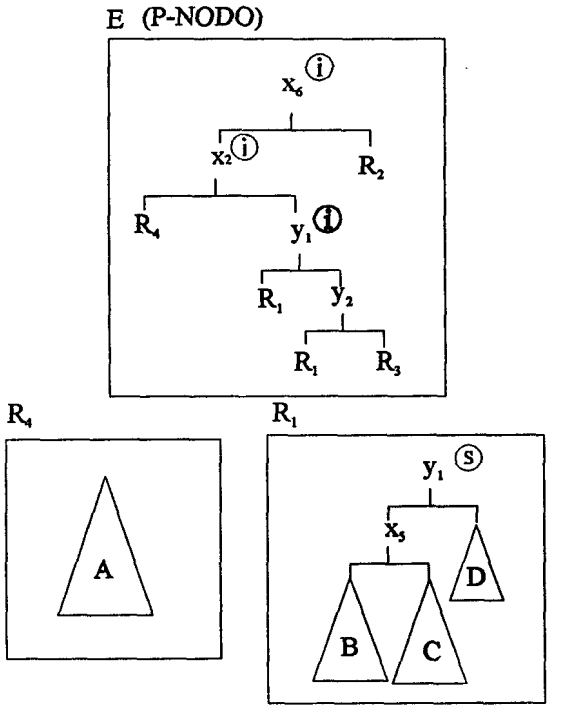


Figura 3-16. Arbol Q resultante después de extraer de R_1 el subárbol A

En el estudio de este caso, varios subcasos pueden presentarse en función de la altura que ocupe la pX-hoja correspondiente al subárbol extraido. Partiendo nuevamente de la figura 3-13, consideremos que la región R_1 está sobrecargada.

Supongamos que se selecciona A como subárbol extraido dentro de R_1. En este caso la pX-hoja situada como hijo izquierdo de x_2 en E, corresponde a la pS-hoja de mayor altura dentro del s-árbol. Por ser así, el nodo local padre de la pX-hoja es un nodo de inclusión. El algoritmo de división de regiones modifica esta pX-hoja para que apunte al nuevo X-NODO R_4. Es claro que **el subárbol hermano de la pX-hoja pasa a ser un nodo de inclusión**. El resultado de la división aparece en la figura 3-16.

Supongamos ahora que seleccionamos como extraido el subárbol enraizado en x_5 dentro de R_1. En este caso, la pX-hoja es la hoja situada en E como hijo izquierdo de y_1. Esta pX-hoja se corresponde con una pS-hoja localizada a una altura media entre otras dos pS-hojas. Por consiguiente, el padre de la pX-hoja no es un nodo de inclusión. El algoritmo de división modifica la pX-hoja por un referencia al nuevo X-NODO R_4. Dado que ni el padre ni el hermano de la pX-hoja son nodos de inclusión, **nada debe ser modificado respecto a las marcas de inclusión**. El resultado de la división aparece en la figura 3-17.

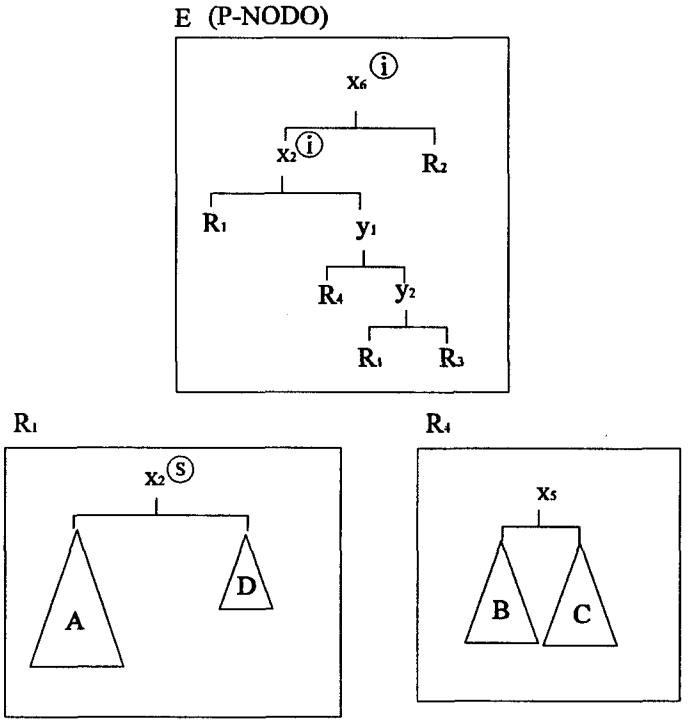


Figura 3-17. Arbol Q resultante tras la extracción del subárbol enraizado en x_5 local a R_1.

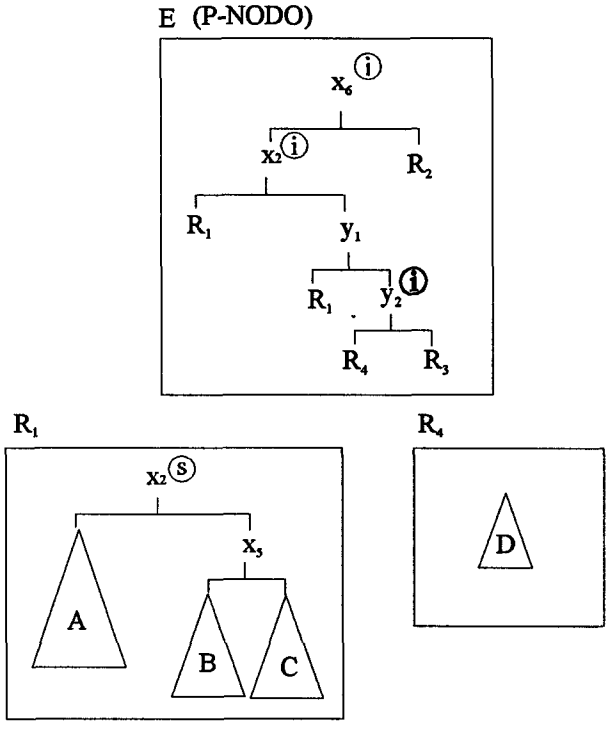


Figura 3-18. Arbol Q resultante tras la extracción del subárbol D local a R_1.

Supongamos por último que, en el árbol de la figura 3-13, seleccionamos D como subárbol para extraer de R_1. En este caso la pX-hoja correspondiente se corresponde con la pS-hoja más profunda de todas las que existen en el p-árbol (hijo izquierdo de y_2 en E). Esto significa que el subárbol hermano de esta pX-hoja o bien es una hoja o es un nodo de inclusión. El algoritmo de división modifica esta pX-hoja para que apunte al nuevo X-NODO R_4. Con esta nueva situación, **el padre de la pX-hoja pasa a ser un nodo de inclusión**. El resultado de la división aparece en la figura 3-18.

Caso 4. La x-raíz posee marca de subido.

Supongamos finalmente que en el marco de la figura 3-13, seleccionamos como extraido el subárbol enraizado en y_1 dentro de R_1. En este caso dado que la x-raíz posee una marca de subido, las pX-hojas son múltiples. Por ser R_1 un Q-nodo no multipadre, de acuerdo a la proposición 3-3, una copia de la x-raíz se encuentra en E. Como todas las pS-hojas descendientes de esta copia en E son modificadas por un apuntador al nuevo X-NODO R_4, **la copia de la x-raíz en el P-NODO pasa a ser un nodo de inclusión**. El resultado de la división aparece en la figura 3-19.

Recordemos que los casos considerados se han estudiado bajo la hipótesis de que el S-NODO no es multipadre. En un escenario en que el S-NODO sea un nodo multipadre, el único caso no aplicable es el caso 4, dado que la copia de la x-raíz puede no localizarse en

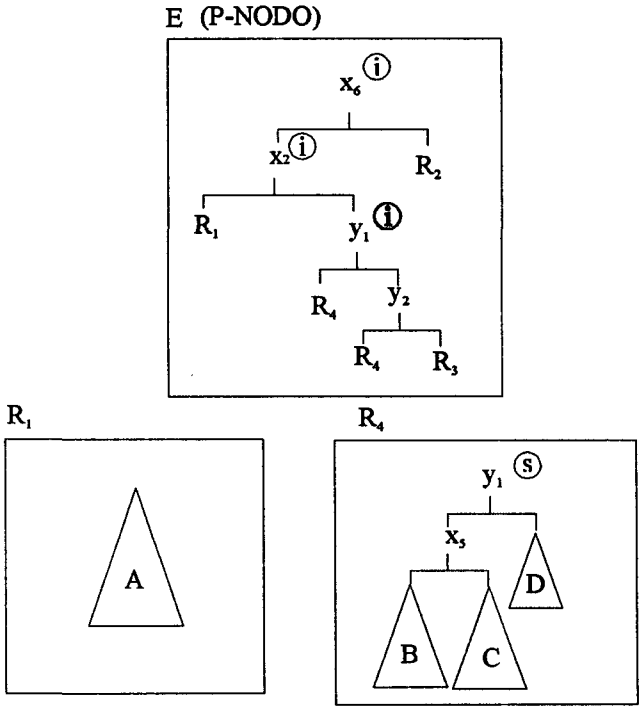


Figura 3-19. Arbol Q resultante tras la extracción del subárbol enraizado en y_1 local a la región R_1.

ninguno de los P-NODOS. En el resto de casos, dado que la pX-hoja es única, un único P-NODO es involucrado en el proceso de colocación de marcas de inclusión, de modo que los tres primeros casos son aplicables sin modificación alguna.

La formalización del método de marcado de nodos de inclusión, ilustrado previamente con la presentación de los distintos casos que pueden aparecer, se recoge en la proposición 3-4, para cuya demostración se precisa el resultado mostrado en el lema 3-3.

Las implicaciones que se deducen a partir de la proposición 3-4, nos revelan la forma en que deben ser marcados los nodos locales del P-NODO en el caso en que una única pX-hoja esté involucrada en el proceso de división del Q-nodo sobrecargado.

Con todo lo anterior podemos ya describir con detalle el método a seguir para el marcado de nodos de inclusión en el proceso de división de Q-nodos, ya sean regiones o espacios de cualquier nivel. Para cada caso, describimos los distintos escenarios que pueden aparecer con sus respectivas soluciones. Son estos:

1. Si la s-raíz no es un nodo subido marcar el primer nodo trasladado al p-árbol como nodo de inclusión.

2. Si la s-raíz es un nodo subido, actuamos en función de la x-raíz.

   2.a Si la x-raíz no es un nodo subido, la pX-hoja es única y el algoritmo de división la modifica para que apunte al X-NODO. Estudiamos el padre de la pX-hoja

      2.a.1 Si el padre de la pX-hoja es nodo de inclusión, marcar el hermano de la pX-hoja como nodo de inclusión

      2.a.2 Si el padre de la pX-hoja no es nodo de inclusión y el hermano de la pX-hoja es una hoja o un nodo de inclusión, marcar el padre de la pX-hoja como nodo de inclusión.

   2.b Si la x-raíz es un nodo subido, actuamos en función de que el S-NODO sea multipadre o no.

Lema 3-3. Nodos de no inclusión

Sea n un nodo de no inclusión y N el árbol con raíz n. No pueden existir dos hojas con distinto valor en N que aparezcan ambas fuera de N.

Demostración.

Supongamos lo contrario. Es decir N posee dos hojas h_1 y h_2 referenciando sendos Q-nodos E_1 y E_2 tales que h_1 y h_2 aparecen también fuera de N.

Por ser n un nodo de no inclusión, algún antecedente de n, es un nodo de inclusión. Sea a el más cercano antecedente de n que es nodo de inclusión y sea A el árbol con raíz a. ⇒

...

Esto significa que A es el más pequeño subárbol que engloba todas las referencias a E_1 y E_2. Según el lema anterior *a* es raíz de E_1 y también raiz de E_2. Pero dos espacios diferentes no pueden nunca tener la misma raíz. Fijémonos que la forma de construir espacios es extrayendo unos a partir de otros y todos los nodos que definen el árbol extraido son movidos a un nuevo Q-nodo. Por tanto la raíz de un árbol extraido no puede ser raíz de otro espacio del mismo nivel. Y lógicamente la raiz de un S-NODO nunca puede ser raíz de un X-NODO, pues esto supondría que todo el S-NODO pasa a ser el X-NODO, lo cual es absurdo. ♦

Proposición 3-4. Marcado de los nodos de inclusión

Supongamos que un Q-nodo, sea multipadre o no, debe ser dividido y que el árbol seleccionado para ser extraido es tal que sus correspondientes pX-hojas son únicas dentro de los P-NODOS del Q-nodo a dividir. Se satisface lo siguiente:

1. Si el nodo padre de una pX-hoja *h* es un nodo de inclusión y dicha pX-hoja es modificada por una nueva referencia *h'*, el subárbol hermano de *h'* pasa a ser un nodo de inclusión, a menos que sea una hoja.
2. Si el nodo padre de una pX-hoja *h* no es nodo de inclusión y esta hoja es modificada por una nueva referencia *h'*, entonces:
   - 2.a Si el hermano de *h* es una hoja, entonces el nodo padre de *h'* pasa a ser un nodo de inclusión.
   - 2.b Si el hermano de *h* es un nodo de inclusión, entonces el nodo padre de *h'* pasa a ser un nodo de inclusión.
   - 2.c Si el hermano de *h* es un nodo local de no inclusión, la situación no varía con la modificación.

Demostración.

Sea *n* el nodo padre de *h*. Llamemos n_1 al hermano de *h*, N al subárbol con raíz *n* y N_1 al subárbol con raíz n_1.

Veamos en primer lugar que si n_1 no es un nodo de inclusión es porque N_1 contiene una copia de *h*. Para demostrar esto supongamos que n_1 no contiene una copia de *h* como hoja. Como por hipótesis n_1 no es nodo de inclusión significa que fuera de N_1 existe alguna hoja h_1 que también se encuentra en N_1. Sin embargo dado que *n* es un nodo de inclusión esta hoja h_1 no puede encontrarse fuera de N. Así pues h_1 es una hoja que se encuentra fuera de N_1 pero dentro de N. Pero en esta situación sólo se encuentra *h*, y por tanto $h_1 = h$.

1. Supongamos que *n* es nodo de inclusión y que la hoja *h* hijo directo de *n* es modificada por *h'*. Si n_1 no es nodo de inclusión, entonces contiene a *h* como hoja. Dado que fuera de N_1 ya no se repite la hoja *h*, n_1 pasa necesariamente a ser un nodo de inclusión.
2. Supongamos ahora que *n* no es un nodo de inclusión. Esto significa que *h* -y no otros valores- aparece dentro y fuera de N.
   - 2.a Si n_1 es una hoja, entonces N es un árbol con dos hojas: *h* y n_1. Si h se modifica por *h'*, *n* pasa a ser un nodo de inclusión, pues dentro de N deja de aparecer *h*.
   - 2.b Si n_1 es un nodo local de no inclusión, por el lema anterior, N_1 contiene al menos una hoja con el mismo valor que *h*. Esto significa que N contiene al menos dos hojas con el valor de *h* y, según la hipótesis, fuera de N también hay hojas con el valor de *h*, luego aunque modifiquemos *h* por *h'*, la situación sigue siendo la misma que antes de la modificación.
   - 2.c Si n_1 es nodo local de inclusión es que N_1 no contiene una copia de *h*, esto es, dentro de N sólo existe una copia de *h*. Si esta es modificada por *h'*, siendo *h'* un valor original, *n* pasa a ser un nodo de inclusión. ♦

2.b.1 Si el S-NODO no es multipadre, marcar la copia de la x-raíz en el p-árbol como nodo de inclusión.

2.b.2 Si el S-NODO es multipadre, retrasamos la descripción hasta el estudio en detalle de este tipo de Q-nodos.

Los distintos pasos de este método serán finalmente integrados convenientemente en el algoritmo de división y traslado de nodos, tanto para regiones como para espacios.

4.3.2 Selección del subárbol extraido

Recordemos los objetivos perseguidos en la selección del subárbol extraido:

1. Cumplir el requisito de carga.
2. Realizar un reparto equilibrado.
3. Disminuir en lo posible el número de referencias redundantes en el Q-nodo padre.
4. Evitar la aparición de Q-nodos multipadre.

En la selección del subárbol extraido para regiones, por no incluir referencias redundantes en su árbol local, el último de los objetivos se consigue de forma automática. Por contra, en el caso de espacios, el último de los objetivos previos toma una relevancia significativa, ya que su consecución significa una simplificación de los algoritmos de división y fusión de Q-nodos.

El apartado anterior nos ha facilitado la información necesaria para conseguir este último objetivo. En efecto, si la extracción de un subárbol dentro de un espacio se realiza por algún nodo de inclusión, resulta evidente la no aparición de Q-nodos multipadre. Teniendo esto en cuenta y sin perder de vista el resto de objetivos, el algoritmo 3-8 describe los pasos a seguir para la selección del subárbol extraido en Q-nodos del conjunto de espacios.

4.3.3 Marcado de Q-nodos multipadre

Cuando en la división de un espacio, el algoritmo de selección de subárbol extraido obtiene como candidato un subárbol cuya raíz no es nodo de inclusión, un Q-nodo descendiente del actual espacio sobrecargado pasa a ser multipadre. Dado que algunas de las acciones anteriormente descritas dependen de que un S-NODO sea o no multipadre, se hace preciso disponer en todo momento de información que nos permita diferenciar Q-nodos multipadre de aquellos que no lo son.
El lema 3-3 nos garantiza en cualquier caso la generación de un único Q-nodo multipadre, todo lo que debemos hacer es encontrar dicho Q-nodo para marcarlo convenientemente.

Algoritmo 3-8. Selección de subárbol extraido para espacios

Entrada: s-árbol

1. $\mathscr{C}$= {Subárboles del s-árbol que cumplen el requisito de carga}. $\mathscr{C} \neq \emptyset$
2. $\mathscr{N}$= {S $\in \mathscr{C}$/ S tiene como raíz un nodo de inclusión}
3. Si $\mathscr{N} \neq \emptyset$, (No generamos Q-nodos multipadre)
4. $\mathscr{S_N}$ = { S $\in \mathscr{N}$/ S tiene como padre un nodo local subido}
5. Si $\mathscr{S_N} \neq \emptyset$ (No conlleva traslado de nodos al P_NODO)
6. Seleccionar S $\in \mathscr{S}$ / FCD(S) = Min (FCD(S_i)), $\forall$ $S_i \in \mathscr{S_N}$
7. Si $\mathscr{S_N} = \emptyset$ (Conlleva traslado de nodos locales al P_NODO)
8. $\mathscr{AMN}$ = {S $\in \mathscr{N}$/ Altura(Raíz de S) = Min(Altura raíz (S_i)), $\forall$ $S_i \in \mathscr{N}$}, (1≤ Card($\mathscr{AMN}$) ≤2)
9. Seleccionar S $\in \mathscr{AMN}$/ FCD(S) = Min (FCD(S_i)), $\forall$ $S_i \in \mathscr{AMN}$
10. Si $\mathscr{N} = \emptyset$, (Generamos la aparición de un Q-nodo multipadre)
11. $\mathscr{S_\mathscr{C}}$ = { S $\in \mathscr{C}$/ S tiene como padre un nodo local subido}
12. Si $\mathscr{S_\mathscr{C}} \neq \emptyset$ (No conlleva traslado de nodos al P_NODO)
13. Seleccionar S $\in \mathscr{S_\mathscr{C}}$ / FCD(S) = Min (FCD(S_i)), $\forall$ $S_i \in \mathscr{S_\mathscr{C}}$
14. Si $\mathscr{S_\mathscr{C}} = \emptyset$ (Conlleva traslado de nodos locales al P_NODO)
15. $\mathscr{AMN}$ = {S $\in \mathscr{C}$/Altura(Raíz de S) = Min(Altura raíz (S_i)), $\forall$ $S_i \in \mathscr{C}$}, (1≤Card($\mathscr{AMN}$) ≤2)
16. Seleccionar S $\in \mathscr{AMN}$/ FCD(S) = Min (FCD(S_i)), $\forall$ $S_i \in \mathscr{AMN}$
17. **Fin.**

Cuando seleccionamos un subárbol con su nodo local raíz como nodo de no inclusión, esto significa que fuera de este subárbol se repiten referencias que aparecen dentro de él. Este subárbol seleccionado puede contener a su vez nodos de inclusión, lo cual significa que las referencias incluidas en los subárboles enraizados en tales nodos no pueden aparecer fuera del subárbol seleccionado. Así pues, las únicas referencias que debemos estudiar son aquellas que se encuentran en el subárbol extraido, descartándo de éste todos los subárboles cuyas raices son nodos de inclusión. Si en el árbol resultante nos encontramos con una única referencia, el Q-nodo al que apunta será necesariamente el Q-nodo multipadre. Si existen varias referencias, bastará con encontrar dos de ellas que se repitan. El caso de que, habiendo más de una referencia no se repitiera ninguna, sólo sería posible si el subárbol extraido contiene un único nodo local y dos hojas. Pero este caso no puede darse si el árbol local al Q-nodo sobrecargado posee 7 o más hojas, ya que el requisito de carga impone una carga mínima de $\lceil h/3 \rceil$, donde h es el número de hojas del árbol local sobrecargado. Teniendo en cuenta el tamaño de los nodos locales y el tamaño de las páginas donde se almacenan los Q-nodos, es seguro que un árbol local sobrecargado posee más de 7 hojas.

Atendiendo a la discusión anterior, el algoritmo 3-9 propone un método para el marcado de Q-nodos multipadre.

Algoritmo 3-9. Marcado de Q-nodos multipadre

Entrada: s-árbol.
Consideraciones: La x-raíz es un nodo de no inclusión.

```
1.  Sea n el nodo actual en consideración. Inicialmente n = x-raíz.
2.  Si alguno de los descendientes de n es una hoja
3.      Acceder al Q-nodo referenciado en dicha hoja y colocarle una marca de multipadre.
4.  si no si alguno de los descendientes de n es un nodo de inclusión
5.      Descender al hijo cuyo descendiente no es nodo de inclusión
6.      Hacer que n apunte al nodo al que acabamos de descender. Ir a 2.
7.  Fin.
```

4.3.4 Identificación de pX-hojas

Siendo ya capaz de distinguir Q-nodos multipadres de aquellos que no lo son, podemos retomar el algoritmo de identificación de pX-hojas descrito para la división de regiones con el fin de plantear un algoritmo genérico que simplifique el proceso de localización de pX-hojas correspondiente al subárbol extraido de un Q-nodo sobrecargado, ya sea una región o un espacio a cualquier nivel.

Como vimos en el apartado 4.2 de división de regiones, cuando la x-raíz del árbol seleccionado como árbol extraido es un nodo no subido, la pX-hoja es única y se encuentra en un único P-NODO. El algoritmo 3-6, en este caso, elige una tupla arbitraria interior a la zona del dominio de búsqueda gobernado por el x-árbol y, posteriormente, recorre el árbol Q en busca de esta tupla para identificar la pX-hoja relevante. Sin embargo, cuando la x-raíz es un nodo subido, las pX-hojas son múltiples y su localización, por aplicación del algoritmo 3-6, requiere acceder a varios contenedores para extraer de cada uno de ellos una tupla con las que realizar el recorrido por el árbol Q para localizar las pX-hojas. Este recorrido del árbol Q ralentiza el proceso de división de Q-nodos.

El subapartado anterior nos proporciona un mecanismo para saber si el Q-nodo a dividir es multipadre o no. En el caso en que el Q-nodo a dividir no sea multipadre, distintas mejoras pueden ser introducidas en el algoritmo 3-6 de identificación de pX-hojas para acelerar el proceso de división.

Un Q-nodo no multipadre sobrecargado posee un único P-NODO, de modo que para acceder al Q-nodo afectado desde la raíz del árbol Q es preceptivo haber visitado previamente el P-NODO. Dado que la inserción de toda tupla en el árbol Q requiere de un recorrido previo por el árbol Q para localizar el contenedor destino, durante este recorrido podemos registrar las direcciones de los Q-nodos visitados, de modo que la localización del P-NODO correspondiente a un Q-nodo no multipadre puede hacerse sin necesidad de recorrer nuevamente el árbol Q.

El algoritmo de identificación de pX-hojas, en el caso en que el S-NODO sea no multipadre, puede mejorarse en los siguientes aspectos:

- Si la x-raíz del árbol seleccionado es un nodo no subido, localizamos una tupla arbitraria en algún contenedor gobernado finalmente por el x-árbol (esta localización requiere bajar por el árbol Q desde el S-NODO hasta un contenedor). Con los datos de esta tupla y dirigiéndonos directamente al P-NODO, cuya dirección es conocida, realizamos un recorrido por el p-árbol hasta bajar a una hoja que apunte al S-NODO. Esta es la pX-hoja. En este caso, nos ahorramos un recorrido desde la raíz del árbol Q hasta el P-NODO.
- Si la x-raíz del árbol seleccionado es un nodo subido, accedemos al P-NODO y realizamos una búsqueda por el p-árbol hasta encontrar allí la copia de la x-raíz. Las hojas del subárbol enraizado en el nodo local encontrado que referencien al S-NODO son las pX-hojas (proposición 3-2). En este caso, nos ahorramos tanto el recorrido desde el S-NODO hacia los contenedores como el recorrido desde la raíz del árbol Q hasta el P-NODO.

Cuando el S-NODO es multipadre, el recorrido previo a la inserción puede no haber visitado Q-nodos que luego se vean implicados como P-NODOS del S-NODO, de modo que, en este caso, resulta necesario atravesar el árbol Q desde su raíz con la (las) tuplas relevantes al x-árbol con objeto de localizar las pX-hojas. Así pues, en el caso poco frecuente de S-NODOS multipadre, el algoritmo 3-6 de identificación de pX-hojas nos conducirá a los nodos locales en Q-nodos del nivel superior implicados en el proceso de división del S-NODO.

Algoritmo 3-10. Identificación definitiva de pX-hojas

Entrada: s-árbol y p-árbol.

1. En el recorrido previo a la inserción de una tupla vamos apilando las direcciones de los Q-nodos visitados, de modo que en cualquier momento tengamos acceso al Q-nodo padre de un S-NODO no multipadre.
2. **si** el S-NODO es un nodo multipadre
3. Aplicar el algoritmo 3-6 para identificar las pX-hojas.
4. **si no si** la x-raíz es un nodo subido
5. Acceder al P-NODO y buscar en el p-árbol la copia de la x-raíz. Todas las hojas descendientes de este nodo local que referencian al S-NODO se identifican como pX-hojas.
6. **si no**
7. Obtener una hoja arbitraria descendiente de la x-raíz y acceder al Q-nodo referenciado
8. **mientras** el Q-nodo accedido no sea un contenedor
9. Obtener una hoja arbitraria del árbol local y acceder al Q-nodo referenciado
10. **fin mientras**
11. Obtener una tupla arbitraria del contenedor accedido
12. Acceder al P-NODO y realizar una búsqueda de esta tupla por el p-árbol hasta alcanzar una hoja. La hoja alcanzada se identifica como pX-hoja
13. **fin si**

Algoritmo 3-11. Marcado de nodos de inclusión

Entrada: s-árbol y p-árbol.

```
1.  si la s-raíz no es un nodo subido
2.      marcar el primer nodo trasladado al p-árbol como nodo de inclusión. Fin.
3.  si no si la x-raíz no es un nodo subido
4.      si el padre de la pX-hoja es un nodo de inclusión
5.          marcar el hermano de la pX-hoja como nodo de inclusión. Fin.
6.      si no si el hermano de la pX-hoja es hoja o nodo de inclusión
7.          marcar el padre de la pX-hoja como nodo de inclusión. Fin.
8.      fin si
9.  si no {La x-raíz es un nodo subido}
10.     localizar el subárbol más pequeño en el (los) P-NODO (S) que engloba todas las
        pX-hojas.
11.     marcar la raíz de este (estos) subárbol (subárboles) como nodo de inclusión (a menos
        que sean hojas)
12. fin si
13. Fin.
```

A partir de la discusión anterior podemos ya escribir el algoritmo definitivo de identificación de pX-hojas, cualquiera que sea el Q-nodo de la zona de índices a dividir.

Los algoritmos 3-10 y 3-11 nos permiten, a falta de especificar el proceso de marcado de nodos de inclusión en el caso de S-NODOS multipadre, ofrecer una visión pormenorizada de las acciones a desarrollar en el proceso de división de Q-nodos y traslado o modificación de nodos locales en Q-nodos del nivel superior. Así pues, antes de describir el algoritmo general de división de Q-nodos, pasemos a estudiar con algo más de detalle las posibles implicaciones de los Q-nodos multipadre en los algoritmos previos.

4.4 Q-nodos multipadre

Los conocimientos adquiridos en los apartados previos sobre el proceso de división de Q-nodos nos capacitan para entender, en toda su extensión, las posibles implicaciones que sobre las acciones involucradas en este proceso tiene el hecho de que el Q-nodo a dividir sea multipadre.

Aunque en el algoritmo de selección del subárbol extraido imponemos como premisa secundaria la no generación de Q-nodos multipadre, puede darse la circunstancia de que ningún subárbol dentro del S-NODO que satisfaga el requisito de carga tenga como raíz un nodo de inclusión, en cuyo caso aparecen Q-nodos multipadre. Debido a ello, es necesario que el proceso de traslado de nodos tenga en cuenta esta posibilidad.

A lo largo de los apartados anteriores hemos ido describiendo paso a paso todas las acciones necesarias para llevar a cabo la división de Q-nodos, ya sean regiones o espacios. Sin embargo, debido al aplazamiento de ciertas propiedades de los Q-nodos multipadre, el algoritmo de marcado de nodos de inclusión en el caso de que el Q-nodo a dividir sea multipadre se retrasó hasta este punto.

La proposición 3-3 nos proporciona el método para colocar marcas de inclusión cuando dentro de un P-NODO (ya sea el S-NODO multipadre o no) o de varios P-NODOS (en el caso de un S-NODO multipadre) la pX-hoja localizada sea única. Además, en virtud del lema 3-3, también sabemos cómo actuar en el caso de una x-raíz subida dentro de un S-NODO no multipadre. Falta por consiguiente establecer el modo de proceder en el caso de un S-NODO multipadre tal que las pX-hojas localizadas se encuentren en distintos P-NODOS y alguno de ellos almacene más de una pX-hoja.

Para ilustrar este caso, consideremos como ejemplo el árbol Q de la figura 3-20. Supongamos que R_1 es una región sobrecargada y que el subárbol seleccionado como subárbol extraido es áquel enraizado en y_1. Las pX-hojas correspondientes dicho subárbol se encuentran en los P-NODOS E_1 y E_2. Además dentro de E_1, las referencias al S-NODO situadas como descendientes de x_5 e y_7 son pX-hojas. Como puede verse en la figura, aunque la x-raíz es un nodo local subido, no existe copia alguna de la x-raíz dentro de ninguno de los P-NODOS. Dado que el procedimiento establecido previamente para marcar nodos de inclusión no contempla una situación así, algún método adicional debe ser empleado para marcar convenientemente los nuevos nodos de inclusión.

Proposición 3-5. Localización de las pX-hojas dentro de un P-NODO

Todas las pX-hojas localizadas en un P-NODO se engloban en un subárbol tal que no contiene otras pS-hojas que las pX-hojas.

Demostración.

Consideremos S el subárbol más pequeño dentro del P-NODO que engloba todas las pX-hojas contenidas en ese P-NODO. Supongamos que S contiene otra pS-hoja no identificada como pX-hoja. Sea *s* la raíz de este subárbol y supongamos que no es copia de la x-raíz, ya que si así fuera, la proposición 3-3 demuestra nuestra afirmación. En ambos subárboles descendientes de *s* se encuentran pX-hojas. Esto significa que el hiperplano que define *s* es interior a la zona del dominio de búsqueda que gobierna el x-árbol y, por consiguiente, el x-árbol incluye a *s* como nodo local subido.

Por una parte y en virtud de nuestras hipótesis, a un lado del hiperplano *s* existe una una zona que el x-árbol no gobierna. Por otra, el hiperplano *s* es interior a la zona extraida y por tanto los puntos que delimita a un lado y otro son interiores al x-árbol. Como es obvio esto esto supone una clara contradicción. ♦

La proposición 3-5 nos allana el camino para establecer el criterio que ha de seguirse a la hora de marcar nodos de inclusión en el caso que nos ocupa.

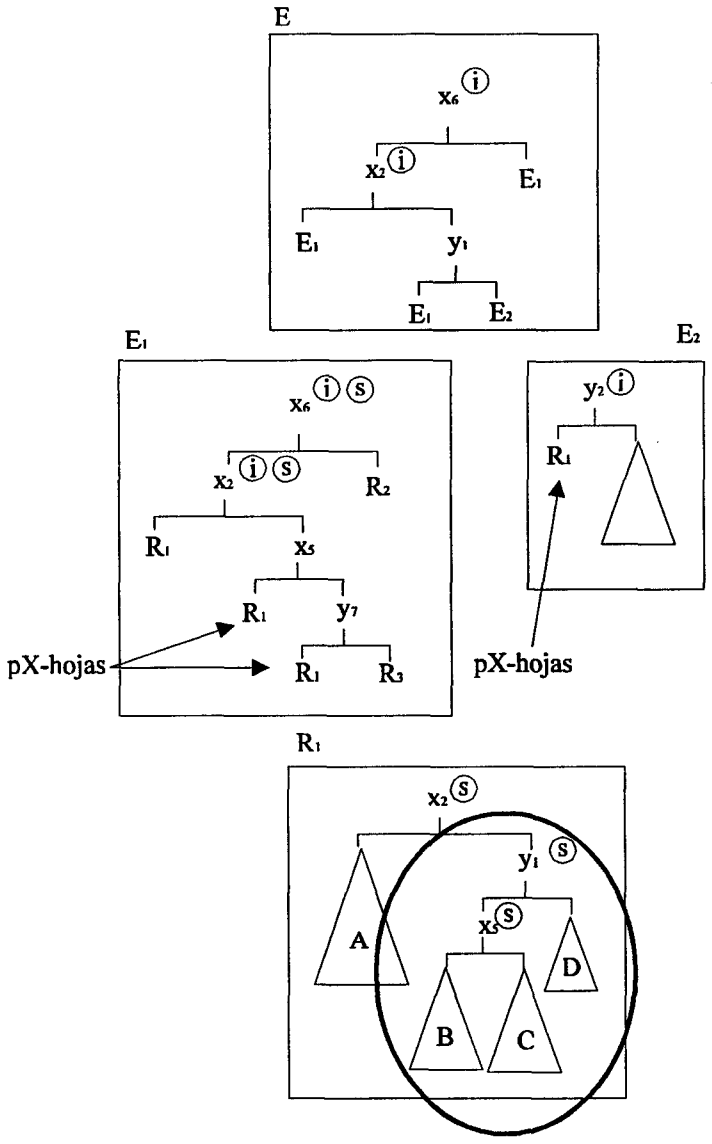


Figura 3-20. Arbol Q con una región R_1 multipadre. Las pX-hojas se localizan en distintos P-NODOS.

En virtud de este resultado, en el caso de un S-NODO multipadre con múltiples pX-hojas en distintos P-NODOS, el marcado de nodos de inclusión debe proceder localizando en primer lugar el subárbol referido en la proposición, para marcar finalmente su raíz como nodo de inclusión.

En la división propuesta como ejemplo en la figura 3-20, la raíz del subárbol más pequeño dentro de E_1 que engloba todas las pX-hojas, esto es el nodo x_5, debe ser marcado como nodo de inclusión. El resultado de la división se muestra en la figura 3-21.

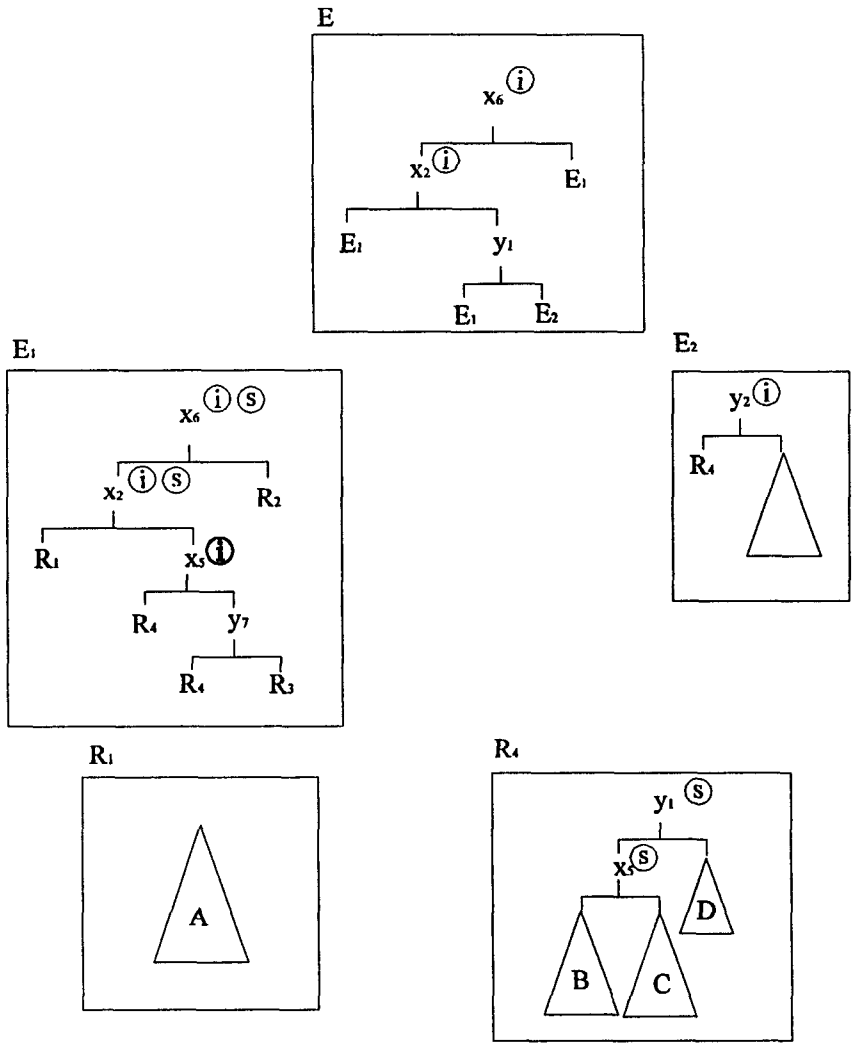


Figura 3-21. Arbol Q resultante tras la división de R_1. El nodo x_5 es finalmente marcado como nodo de inclusión.

Una vez resuelto el marcado de nodos de inclusión en el caso de S-NODOS multipadre, podemos finalizar este apartado aportando el algoritmo definitivo de marcado de nodos de inclusión antes de pasar a la descripción del algoritmo genérico de división de Q-nodos. Los detalles relativos al mismo se muestran en el algoritmo 3-11.

5. Algoritmo definitivo de división de Q-nodos

El apartado anterior nos ha ofrecido una visión detallada y por partes sobre los diferentes aspectos que intervienen en el proceso de división de Q-nodos. Ello nos ha permitido entender en toda su amplitud el papel que, dentro de la organización global del espacio de búsqueda, juegan los Q-nodos a diferentes niveles en el árbol Q. En el presente apartado aportamos una visión globalizadora del proceso de división.

Algoritmo 3-12. División de un S-NODO

Entrada: S-NODO.

```
si el S-NODO es un contenedor
    asignar un nuevo contenedor, repartir el contenido, construir un nodo local con el valor
    mediana sobre el atributo elegido para la división e insertar en el P-NODO este nuevo
    nodo local. El hijo izquierdo apunta al antiguo contenedor y el hijo derecho al nuevo
    contenedor.
fin si
si el S-NODO es una región
    aplicar el algoritmo 3-3 de selección de subárbol
    copiar el subárbol extraido sobre una nueva región
    aplicar el algoritmo 3-10 para identificar las pX-hojas
    aplicar el algoritmo 3-7 de traslado de nodos o modificación de pX-hojas
    modificar convenientemente el S-NODO para convertirlo en el MS-NODO
fin si
si el S-NODO es un espacio
    aplicar el algoritmo 3-8 de selección de subárbol extraido
    copiar el subárbol extraido sobre un nuevo espacio adjuntado al árbol Q
    si la x-raíz no es nodo de inclusión
        aplicar el algoritmo 3-9 para marcar un descendiente como Q-nodo multipadre
    fin si
    aplicar el algoritmo 3-10 para identificar las pX-hojas
    si la el padre de la x-raíz no es un nodo subido
        aplicar el algoritmo 3-7 de traslado de nodos, marcando el primer nodo subido como
        nodo de inclusión
    si no
        modificar las pX-hojas para que apunten ahora al X-NODO
        aplicar el algoritmo 3-11 de marcado de nodos de inclusión
    fin si
    modificar convenientemente el S-NODO para convertirlo en el MS-NODO
fin si
```

El algoritmo de división que proponemos se encuentra a un alto nivel de abstracción, por lo que en alguno de sus puntos podría parecer ineficiente. Es por ello que el algoritmo aquí descrito ha de ser interpretado en sus justos términos como simple portador de una perspectiva global del proceso de división de Q-nodos.

Los detalles específicos del mismo se muestran enmarcados bajo el epígrafe del algoritmo 3-12.

6. Conclusiones

En el presente capítulo hemos presentado tanto los aspectos estructurales, como los algoritmos de creación de un nuevo método de indexación multiatributo denominado árbol Q.

El árbol Q constituye un mecanismo adaptativo para la recuperación de registros sobre la base de múltiples atributos que cubre con eficiencia todas las clases de búsqueda de puntos en un dominio multidimensional. Aunque en algunos aspectos, el árbol Q incorpora ideas prestadas de la estructura de árbol hB, tales como el uso de árboles k-d como estructura interna para las páginas del índice y, en cierta forma, el mecanismo de división de páginas índice mediante la extracción de subárboles, la filosofía global implícita en la organización del espacio de búsqueda, así como los mecanismos específicos de división y traslado de términos índices hacia niveles superiores del árbol Q, son completamente originales y aparecen por primera vez en la presente tesis.

Básicamente, las diferencias respecto del árbol hB pueden resumirse en los siguientes puntos.

- *El uso de la clave primaria como atributo comodín en el particionamiento del espacio de búsqueda.* Esta característica permite realizar, a diferencia de otros métodos de acceso multiatributo, una división uniforme del espacio bajo cualquier circunstancia. Para establecer con mayor claridad la importancia de tal atributo, consideremos una relación con n atributos, de los cuales $k$de ellos desean ser tratados como claves secundarias ($k < n$). Supongamos que entre estos $k$atributos no se encuentra una clave primaria. En el momento de la división de una página de datos, según el mecanismo establecido en el árbol hB, tomando a lo sumo los valores de dichos k atributos, es posible realizar la división con un ratio garantizado de 2:1. Sin embargo, en el escenario propuesto, una división así no puede ser garantizada en todo momento. Basta considerar el caso extremo de que todas las tuplas en una página de datos tengan los mismos valores sobre los k atributos claves secundarias de la relación, situación perfectamente posible habiendo considerado que ninguno de esos k atributos identifica de forma única una tupla de la relación.
- *El factor de ocupación de las páginas de datos puede ser controlado por el usuario*, no dejando al azar, o mejor dicho, a la distribución que siguen los valores tomados por las tuplas sobre los distintos atributos, la decisión sobre el ratio de ocupación tras la división de un contenedor. La participación de la clave primaria como atributo adicional de división garantiza esta propiedad.
- *El árbol Q no precisa mantener información frontera en cada uno de sus Q-nodos índice*, evitando así la ocupación de una cantidad considerable de espacio de almacenamiento en el caso de indexar un número considerable de atributos de la relación.
- *Los algoritmos de selección de subárboles* en la división de nodos índice, no sólo persiguen la consecución de reparto uniforme del contenido, sino que van más allá, incorporando facilidades que simplifican las posteriores operaciones de traslado de nodos hacia niveles superiores, disminuyendo en lo posible la información trasladada y evitanto, por consiguiente, un crecimiento desmesurado de la estructura.

Además de lo anterior, la metodología de construcción del árbol Q lleva implícita una organización del espacio de búsqueda capaz de adaptarse de forma natural a las posibles fluctuaciones respecto del número de tuplas que alberga el fichero indexado. En este sentido, en la definición de los algoritmos de división y ascenso, hemos cuidado la metodología seguida con objeto de hacer factible su reversibilidad ante la operación del borrado de tuplas. Así por ejemplo, mientras que en la estructura de árbol hB, durante el proceso de traslado de términos índice pueden perderse nodos locales (en principio redundantes e inútiles para la búsqueda) que describen la frontera de la región extraida, en el árbol Q hemos optado por mantener viva en todo momento esta información frontera sin la cual, un intento de deshacer la división previa, concatenando nuevamente regiones divididas, hace impracticable una vuelta a la situación anterior a la división.

Debido al coste temporal que ha supuesto la implementación de nuestra estructura de árbol Q, su posterior evaluación y aplicación final como método básico soporte del particionamiento multidimensional de relaciones, la descripción definitiva de los algoritmos de borrado, así como su instrumentalización como parte de nuestro sistema de indexación mediante árboles Q, han sido postergadas para su exposición en un trabajo futuro.

El establecimiento definitivo de un método de indexación multiatributo en bases de datos de propósito general requiere un serio trabajo de implementación, análisis y evaluación que ofrezca un mecanismo de retroalimentación capaz de reportar información precisa para su aplicación, libre de errores en los ámbitos para los cuales está pensado. Si bien podemos afirmar que una parte importante de este trabajo ha sido realizado, mucha dedicación resulta todavía necesaria hasta conseguir resultados óptimos. En este sentido, los trabajos que en la actualidad está siendo desarrollados y otros que serán objeto de dedicación en un futuro inmediato se centran en los siguientes aspectos:

- Definición e implementación de algoritmos de borrado en el árbol Q. Aunque ya disponemos de versiones iniciales sobre los mismos, la implantación de tales algoritmos y su adaptación a la versión más reciente de nuestro árbol Q está siendo en la actualidad objeto de desarrollo por parte de nuestro grupo de trabajo
- Implementación de una estructura alternativa para la representación de Q-nodos del conjunto de espacios, basados en la utilización de árboles binarios cuyos nodos internos representan la frontera multidimensional de las regiones extraidas. Su comparación con la versión actual bajo distintos conjuntos de datos nos aportará información valiosa sobre el comportamiento de ambas alternativas bajo diferentes escenarios.
- Aplicación del árbol Q para la indexación de datos espaciales. Tal y como argumentamos en el capítulo 2, los métodos de indexación multiatributo ofrecen posibilidades de aplicación en entornos espaciales, donde los objetos a buscar no son necesariamente puntos en el espacio sino que tienen representaciones diferentes, tales como regiones multiformes, figuras geométricas, etc. Estas posibilidades van a ser inspeccionadas en el marco de nuestro grupo de investigación.

- Construcción de una batería completa de pruebas orientada al estudio del rendimiento de estructuras índice multiatributo. Esta batería nos permitirá en un futuro próximo comparar con criterios objetivos el rendimiento de las diferentes estructuras de indexación multiatributo, con objeto de valorar más precisamente la utilidad, ventajas e inconvenientes del árbol Q sobre métodos similares de acceso multiatributo. La inexistencia de "*benchmarks*" orientados a la evaluación de estas estructuras, supone un serio obstáculo para el desarrollo de tales estructuras. Creemos que un serio estudio en este ámbito podría aportar herramientas muy valiosas para los investigadores en este campo de aplicación.

Capítulo 4

Particionamiento multidimensional basado en el árbol Q

Para explotar el paralelismo en las bases de datos, es necesario dividir las consultas en múltiples tareas que puedan ser ejecutadas en paralelo. La efectividad del procesamiento de operaciones relacionales en sistemas paralelos depende en gran medida de la habilidad de distribuir la carga correspondiente a cada una de las tareas equilibradamente entre todos los nodos de procesamiento [51, 41, 73, 47].

La labor de asignación equilibrada de la carga entre los procesadores no resulta excesivamente compleja en sistemas de memoria compartida, donde cada procesador puede acceder a cualquier porción de los datos. Sin embargo, en sistemas de memoria distribuida, donde un procesador no tiene acceso directo a la memoria o discos de los otros procesadores, el objetivo de equilibrar la carga entre los nodos de procesamiento resulta ser crítico en la consecución de unos niveles aceptables de rendimiento global del sistema.

Ya que, por lo general, el componente de entrada/salida de datos supone la principal aportación en el tiempo global de ejecución de una consulta, el equilibrio de carga en la distribución de tareas relativas a una consulta pasa necesariamente por una distribución equilibrada de los datos de entrada a la consulta.

Si consideramos una consulta consistente en un único operador de selección ("*Select*"), la anterior valoración no merece discusión alguna. Por contra, en el caso frecuente de una consulta consistente en un arbol de operadores relacionales, donde la salida de un operador es la entrada para el siguiente, se podría argüir que una distribución inicial equilibrada de los datos de entrada aportará equilibrio sólo en la primera parte de la consulta (hojas del árbol de operadores), ya que la distribución de los datos de entrada puede ser completamente diferente de la de los datos de salida. Sin embargo, dado que en el plan de ejecución de una consulta relacional, la operación de selección se realiza en los estados iniciales de la consulta, siendo además de uso muy frecuente, resulta indiscutible la utilidad de un mecanismo de distribución inicial de datos que, al menos, no hipoteque en sus inicios la ejecución eficiente de la consulta en una máquina paralela.

Aunque existe una amplia problemática en torno al establecimiento de mecanismos de equilibrio de carga en operaciones de concatenación ("*Join*") [47], nuestros trabajos se alejan de este campo. En tal sentido, en el seno de nuestro grupo de investigación, actualmente se desarrollan trabajos enmarcados en este ámbito, cuyos avanzados resultados son objeto de presentación de una nueva tesis doctoral en un futuro muy próximo.

El principal inconveniente para conseguir una distribución equilibrada de los datos de entrada a la consulta, radica en la posibilidad realista de que ciertos valores de atributo en una relación ocurran más frecuentemente que otros. Esta característica de oblicuidad es inherente a los propios datos y no depende en absoluto del modelo de acceso a los mismos, lo que provoca que, en presencia de sesgo en los datos, la habilidad de distribuir uniformemente la carga entre los procesadores pueda verse seriamente limitada.

El estudio de mecanismos de particionamiento de relaciones capaces de aportar una solución eficiente al problema del equilibrio de carga es el objeto del presente capítulo. A lo largo de este capítulo identificamos la problemática expuesta, analizamos las diferentes propuestas para su resolución y presentamos, finalmente, una propuesta original de particionamiento multidimensional de los datos entre los nodos de una máquina multicomputador de memoria distribuida. Este mecanismo de particionamiento multidimensional de relaciones hace uso del método de acceso indexado en árbol Q, cuya descripción fue objeto de atención en el capítulo anterior. Además de todo lo anterior, ampliamos el concepto de explotación del índice en entornos paralelos presentando las ideas básicas sobre las que actualmente trabajamos. En este sentido, no sólo nos preocupamos del particionamiento de los datos, sino que atendemos también a la paralelización del índice en sí mismo.

Para presentar estos contenidos, este capítulo se estructura del siguiente modo. El apartado 1 enuncia la motivación que nos conduce a presentar nuestra propuesta de particionamiento basado en el árbol Q. El apartado 2 describe en términos generales la arquitectura multicomputador sobre la que basaremos nuestras propuestas de particionamiento. Previo a su descripción, el apartado 3 realiza un breve recorrido por las principales ideas y el estado actual sobre los distintos mecanismos de particionamiento multidimensional establecidos

hasta la fecha. Los diferentes enfoques que proponemos para el particionamiento multidimensional basado en el árbol Q se exponen en el apartado 4. Con el fin de evaluar posteriormente el comportamiento de nuestros mecanismos de particionamiento en entornos paralelos y obtener resultados comparativos frente a enfoques más clásicos de particionamiento, el apartado 5 define el modelo analítico de simulación utilizado para medir la ejecución de las consultas. El apartado 6 avanza las ideas sobre distribución y uso del índice árbol Q en entornos paralelos. Finalmente, el apartado 7 presenta las principales conclusiones y lineas de trabajo futuro respecto a la utilización del árbol Q en sistemas paralelos.

1 Explotación del árbol Q en sistemas paralelos

Dos son las motivaciones que nos han llevado a estudiar en profundidad la forma de explotar el índice en árbol Q sobre sistemas paralelos:

- Integración de métodos de acceso en la ejecución paralela de consultas.
- Problemática de los mecanismos tradicionales de particionamiento ante situaciones críticas de desequilibrio de carga.

Respecto al primer punto, la reflexión que hace Lu [47] nos sirve para ilustrar la primera de las motivaciones expuestas:

> *"As with conventional databases, indexes drastically affect the performance of the parallel processing algorithms. However, unlike conventional databases, instead of a class of access methods, most of the research in parallel query processing considers only hash-based methods. The use of only one access method has the disadvantage of not being able to exploit access patterns and data characteristics. Furthermore, the support of more than one index provides more space for optimization. [...] A main reason for not supporting sophisticated indexes is the complexity of maintaining and making use of indexes when data are stored at and processed by more than one processor."*

La anterior reflexión nos induce a pensar en la necesidad de abrir nuevas vías de indexación que puedan ser aplicables en el proceso de ejecución paralela de consultas.

En cuanto a la problemática del desequilibrio, la mayoría de las bases de datos paralelas utilizan técnicas sencillas de particionamiento de relaciones basadas en el uso de un atributo de la relación (típicamente la clave primaria). Estas técnicas provocan una falta de uniformidad en el tamaño de las subrelaciones cuando los valores del atributo de particionamiento muestran una distribución sesgada, lo que conduce a un desequilibrio de carga en la operación (típicamente temprana) de selección de tuplas. Además, cuando en el predicado de la consulta aparecen atributos distintos del atributo de particionamiento, la ejecución de la consulta debe enviarse a todos los procesadores en cuyo espacio de almacenamiento alberguen algún fragmento de la relación.

La primera de las motivaciones expuestas supone una vía de investigación que es objeto de continuación en nuestras líneas de trabajo futuro. Como se desprende del capítulo anterior, el árbol Q es una estructura indexada altamente susceptible de paralelización en sí misma. Las operaciones de búsqueda en el árbol conllevan un recorrido que se bifurca, a partir de una página del índice, a varias páginas de nivel inferior. Una distribución apropiada de las páginas del árbol entre múltiples procesadores sugiere, de forma natural, la paralelización del algoritmo de búsqueda en el índice, así como de las operaciones de mantenimiento del mismo. Las ideas básicas sobre las distintas vías de paralelización que actualmente investigamos se exponen brevemente al final del capítulo.

Aparte de esto, el capítulo actual describe básicamente un conjunto de técnicas destinadas a resolver la problemática que se deriva de la segunda de las motivaciones expuestas. En este sentido, a lo largo del mismo presentaremos un esquema original de particionamiento multidimensional como propuesta general al particionamiento de relaciones en el ámbito de las bases de datos paralelas.

Es conveniente resaltar que el trabajo experimental desarrollado con objeto de evaluar nuestra propuesta, se centra con exclusividad en el análisis de la operación de selección sobre una relación. La explotación sobre sistemas paralelos de índices multiatributo similares al árbol Q (más específicamente una generalización del árbol Q, llamada árbol mQ) en el conjunto de operadores multitupla constituye un trabajo de investigación desarrollado en el seno de nuestro grupo por otro de los componentes del mismo, el cual es objeto de presentación de una nueva tesis doctoral que de alguna forma complementa y extiende los trabajos que aquí presentamos.

Antes de entrar en detalle sobre las propuestas de particionamiento, pasamos a describir las lineas básicas de la arquitectura de la máquina sobre la que desarrollamos nuestras ideas acerca del modelo de particionamiento basado en el árbol Q.

2 El modelo arquitectural

Tal y como se introdujo en el capítulo 1, el trabajo desarrollado en la presente tesis viene motivado por la necesidad de aportar un mecanismo de particionamiento sobre el cual poder asentar la ejecución de transacciones en la máquina paralela de base de datos QUATRO. En este apartado describimos a grandes rasgos el modelo arquitectural de QUATRO, sobre el cual construimos nuestro modelo de ejecución de consultas, utilizando los diferentes esquemas de particionamiento propuestos.

Hoy en día es prácticamente un consenso entre la comunidad investigadora la necesidad de utilizar una arquitectura multicomputador de memoria distribuida si se desea alcanzar el objetivo de escalabilidad del sistema. Las arquitecturas de memoria distribuida disminuyen la interferencia dentro del sistema, reduciendo la compartición de recursos. Tales arquitecturas

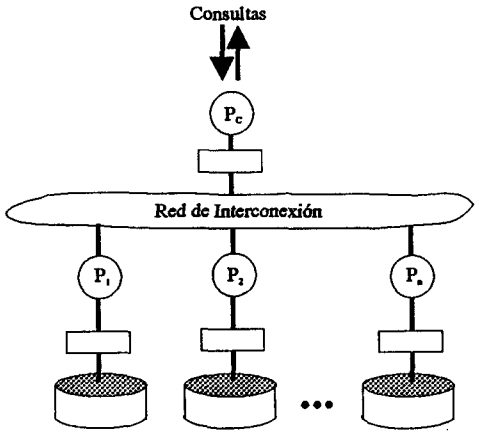


Figura 4-1. Modelo arquitectural de la máquina QUATRO

explotan los recursos de procesador y memoria sin necesidad de una red de interconexión increíblemente potente. El diseño de memoria distribuida mueve a través de la red exclusivamente solicitudes y respuestas, de modo que permite un diseño más escalable disminuyendo el tráfico sobre la red de interconexión.

De acuerdo a los planteamientos expuestos en el capítulo 2, la máquina de base de datos QUATRO se diseña sobre un sistema multicomputador de memoria distribuida, donde cada procesador dispone de memoria privada y algunos procesadores tienen acceso directo a un disco.

El modelo de ejecución que nosotros planteamos considera la existencia de un procesador de control cuya misión es interactuar con el exterior así como controlar y coordinar las actividades de los distintos procesadores en la ejecución de una consulta.

La figura 4-1 muestra un esquema de la arquitectura multicomputador sobre la que construimos el modelo de ejecución de consultas.

Si bien el procesador de control (P_C en la figura 4-1) no realiza manipulación directa de datos, sí tiene acceso a información de catálogo, directorio y, en general, cualquier información metadato de interés para planear la ejecución de la consulta.

Con esta arquitectura de memoria distribuida, en la cual los procesadores comunican entre sí por paso de mensajes a través de la red de interconexión, las tuplas de cada relación de la base de datos se distribuyen entre las unidades de disco conectadas directamente a cada procesador. Como se dijo en el capítulo 2, el particionamiento de relaciones permite explotar el ancho de banda de la entrada/salida de múltiples discos convencionales leyendo y escribiendo en paralelo, lo cual evita la inversión costosa en caros dispositivos de entrada/salida provistos de una circuitería especializada tales como discos RAID [58]. En la arquitectura propuesta, se puede decir entonces que cada procesador "gobierna" directamente los datos asignados a "su" disco local.

Nuestro diseño de memoria distribuida permite explotar dos formas de paralelismo en la ejecución de consultas: el paralelismo *intraconsulta* y el paralelismo *interconsulta*.

El paralelismo **intraconsulta** se consigue paralelizando una consulta y llamando a ejecución a aquellos procesadores que "gobiernan" datos de entrada a la consulta. Por su parte el paralelismo **interconsulta** supone aprovechar aquellos procesadores "dormidos" durante la ejecución de una consulta para ejecutar tareas pertenecientes a otra consulta distinta. Si el mecanismo de particionamiento elegido es capaz de discernir a priori qué procesadores son susceptibles de "gobernar" datos relativos a una consulta (aquellos que satisfacen el predicado de selección), sólo tales procesadores deberían ser llamados a participar en la ejecución de la consulta, liberando al resto de procesadores del sistema para poder ejecutar tareas relativas a una consulta diferente.

Se concluye de lo anterior que el mecanismo de particionamiento de relaciones debería considerar en su filosofía la arquitectura subyacente con objeto de explotar en toda su extensión las potencialidades del paralelismo.

3 Modelos de particionamiento multidimensional

A pesar de la existencia de una gran cantidad de estudios previos dedicados al problema del particionamiento en bases de datos paralelas, sólo algunos de ellos encaran la problematica que suscita este aspecto del paralelismo desde la perspectiva multidimensional.

Existen básicamente dos lineas diferentes de trabajo que han enfocado la problemática del particionamiento en bases de datos paralelas desde el concepto multidimensional:

- Mecanismos de distribución de registros basados en el concepto de ficheros de producto cartesiano.
- Particionamiento del espacio de tuplas por subdivisión en rangos de los distintos dominios correspondientes a los atributos del fichero.

Respecto al primero de los enfoques expuestos previamente, existen varios métodos de asignación, de entre los cuales podemos destacar como más representativos aquellos que se proponen en [1, 21, 38]. En [21] se proponen métodos de asignación de páginas basadas en la función módulo. Abdel-Ghaffar y El Abbadi estudian en [1] condiciones necesarias y suficientes de optimalidad de los métodos de asignación de páginas mediante las propiedades implícitas en la teoría de la codificación. Por su parte, en [38] Kim y Pramanik proponen un método de asignación denominada distribución FX ("*fieldwise exclusive*").

En cuanto a la tendencia apuntada en el segundo de los puntos, la práctica totalidad de mecanismos de particionamiento multidimensional propuestos [27, 57, 63] utilizan alguna forma de directorio "rejilla", el cual es utilizado en la fase final de asignación para distribuir las celdas entre los distintos discos.

Los siguientes subapartados introducen las ideas que subyacen en los métodos mencionados.

3.1 Distribución de ficheros producto cartesiano.

Los trabajos centrados en este ámbito plantean en su mayoría el diseño de un sistema de ficheros que aporta cierto grado de paralelismo en el acceso a páginas de datos cuyas tuplas satisfacen una consulta de coincidencia parcial.

Suponiendo que una página puede ser accedida sobre una unidad de disco en una unidad de tiempo, resulta obvio que varias páginas pueden accederse en una unidad de tiempo si están localizadas en discos diferentes. En este caso, el tiempo de respuesta a una consulta de coincidencia parcial ya no es proporcional al número total de páginas que deben examinarse, sino al número máximo de páginas que se acceden sobre una unidad de disco particular.

De acuerdo a este planteamiento, los trabajos relativos al particionamiento de ficheros producto cartesiano se centran en el siguiente objetivo: dado un fichero multiatributo y un conjunto de m unidades de disco, encontrar la forma de asignar a los discos las páginas del fichero de modo que, para cualquier consulta de coincidencia parcial, el tiempo de respuesta sea mínimo.

Con el fin de poner de manifiesto la interesante relación que existe entre ficheros producto cartesiano y la respuesta a consultas de coincidencia parcial, definimos a continuación el concepto de fichero producto cartesiano

Sea D_i el dominio del i-ésimo atributo de un fichero con k atributos y consideremos D_i fragmentado en m_i subconjuntos disjuntos D_{i1}, ...,D_{im_i}. Se dice que F es un fichero producto cartesiano si todos los registros de una página cualquiera, pertenecen al conjunto $D_{1i_1} \times D_{2i_2} \times ... \times D_{ki_k}$, donde cada D_{ji_j} es uno de los subconjuntos D_{j1}, ..., D_{jm_j}.

Para facilitar la comprensión de la definición anterior consideremos como ejemplo un fichero F con atributos X e Y. Sean $D_X = D_Y = \{a,b,c,d\}$, $D_{X_1} = D_{Y_1} = \{a,b\}$ y $D_{X_2} = D_{Y_2} = \{c,d\}$. Supongamos que cada par perteneciente a $D_X \times D_Y$ es una tupla de F. Consideremos ahora dos posibles configuraciones para nuestro fichero, que denominaremos F_1 y F_2. En ambas configuraciones, las tuplas se distribuyen entre cuatro páginas, tal y como muestra la tabla 4-1.

Tabla 4-1. Distribución de tuplas en las dos configuraciones del fichero *F*

Página nº	Tuplas en F_1	Tuplas en F_2
1	(a,a),(a,b), (b,a), (b,b)	(a,a), (b,b), (c,c), (d,d)
2	(a,c), (a,d), (b,c), (b,d)	(a,b), (b,c), (c,d), (d,a)
3	(c,a), (c,b), (d,a), (d,b)	(a,c), (b,d), (c,a), (d,b)
4	(c,c), (c,d), (d,c), (d,d)	(a,d), (b,a), (c,b), (d,c)

Como puede observarse a partir de la tabla 4-2, la configuración F_1, que corresponde a un fichero producto cartesiano, consigue reducir sensiblemente el número de accesos ante las distintas consultas de coincidencia parcial, respecto de la configuración F_2, cuya distribución de tuplas no satisface los requisitos necesarios para ser considerado un fichero de producto cartesiano.

Si, adicionalmente, es posible tener en cuenta la frecuencia con la que se emiten las consultas de coincidencia parcial sobre el fichero de producto cartesiano, un particionamiento ciudadoso de los dominios y una asignación acorde de las páginas entre los distintos dispositivos de disco, permite reducir sensiblemente el tiempo de respuesta a la mayoría de las consultas emitidas.

La forma de asignar las páginas entre los distintos discos es, de hecho, la característica que distingue los diferentes métodos que se acogen a esta filosofía. Por poner un ejemplo, en el caso del método de asignación basado en la función módulo, conocido como "*disk modulo allocation method*" [21], una página del fichero producto cartesiano conteniendo tuplas de $D_{1i_1} \times D_{2i_2} \times ... \times D_{ki_k}$ se asigna a la unidad de disco $(i_1 + i_2 + ... + i_k)$ *mod m*, siendo *m* el número de unidades de disco.

Tabla 4-2. Acceso a las páginas en F_1 y F_2 según la consulta de coincidencia parcial

Consulta	Páginas en F_1	Páginas en F_2
(X = a, Y = *)	1,2	1,2,3,4
(X = b, Y = *)	1,2	1,2,3,4
(X = c, Y = *)	3,4	1,2,3,4
(X = d, Y = *)	3,4	1,2,3,4
(X = *, Y = a)	1,3	1,2,3,4
(X = *, Y = b)	1,3	1,2,3,4
(X = *, Y = c)	2,4	1,2,3,4
(X = *, Y = d)	2,4	1,2,3,4

A pesar de las indudables ventajas que muestran los métodos de particionamiento basados en el concepto de ficheros de producto cartesiano respecto a la disminución del tiempo de respuesta ante consultas de coincidencia parcial, estos métodos sufren de dos serios inconvenientes:

- Nula eficacia ante consultas de tipo rango, las cuales aparecen con cierta frecuencia sobre determinados atributos en bases de datos de propósito general.
- Problemas de desequilibrio en la distribución de tuplas ante la presencia de sesgo en los datos. La falta de uniformidad en la frecuencia con la que aparecen los valores del dominio en las tuplas de la relación puede provocar grandes diferencias de ocupación entre unas páginas y otras y, en consecuencia, la asignación de un volumen muy diferente de datos entre distintas unidades de disco.

Estas desventajas nos conducen a desestimar este tipo de métodos de particionamiento para nuestra máquina QUATRO.

3.2 Particionamiento por sudivisión en rango de los dominios

La filosofía que subyace en este tipo de particionamiento consiste en dividir el espacio de tuplas por múltiples dominios de atributo con el fin de construir un directorio de tipo "rejilla" que, a la postre, referencia un conjunto de celdas conteniendo cada una determinado número de tuplas. En la fase final del proceso, estas celdas se asignan de diferente modo (según el enfoque seguido) a un conjunto de unidades de disco o procesadores.

Los trabajos más representativos de este tipo de técnicas de particionamiento se localizan en las referencias [27, 34, 42 y 57]. Aunque los métodos presentados en todos estos trabajos son susceptibles de utilizarse como mecanismos de particionamiento multidimensional, aquellos que se presentan en [34 y 57], persiguen fundamentalmente minimizar el impacto de la distribución de datos durante la construcción de una tabla de desmenuzamiento ("*hash*") en la ejecución de una operación de join desmenuzado paralelo ("*parallel hash join*").

En los casos de [27 y 42] se proponen sendos mecanismos verdaderamente orientados al particionamiento multidimensional de relaciones.

Li, Srivastava y Rotem definen en [42] el método conocido como CMD ("*Coordinate Modulo Distribution*"), en el cual cada dimensión del espacio de tuplas se divide en nM intervalos, cada uno de longitud $1/nM$, donde M es el número de discos y n es un parámetro del particionamiento. De este modo un espacio d-dimensional se puede representar por medio de un directorio en forma de rejilla, en donde cada celda con coordenadas $(X_1, X_2, ..., X_d)$ se asigna, en la última fase del proceso, al disco $(X_1 + X_2 + ... + X_d)$ *mod* M.

Aunque los análisis de rendimiento llevados a cabo en [42] determinan un buen comportamiento del método CMD frente a consultas de tipo rango, se hecha en falta una evaluación más rigurosa de este método en situaciones realistas. Así por ejemplo, en los resultados de similación del método se presupone distribución uniforme de los valores sobre cada uno de los dominios de atributos de la relación. De modo similar, la carga de trabajo a la que se somete el sistema se compone de consultas de rango aleatoriamente generadas, sin consideraciones alternativas de posible sesgo en su generación. Adicionalmente, el comportamiento del método CMD ante consultas en cuyo predicado participen sólo una porción de los atributos de particionamiento resulta ser una verdadera incógnita.

Por su parte, Ghandeharizadeh y DeWitt establecen en [27] un mecanismo de particionamiento llamado MAGIC, cuyo planteamiento está más estrechamente relacionado con nuestros objetivos. La estrategia de particionamiento MAGIC, consiste en la creación de un directorio "rejilla" construido sobre los atributos de particionamiento indicados por el usuario. Para ello es preciso conocer los requerimientos de cada operación de selección sobre una relación y su frecuencia de ejecución. Con esta información, MAGIC establece el número óptimo de procesadores que deben intervenir en la ejecución de una consulta promedio. Además de ello y en función de la frecuencia con que los distintos atributos aparecen en el predicado de las consultas, se calcula el número de procesadores que debería intervenir en la ejecución de consultas cuyo predicado incluya un determinado atributo. Con tales datos, MAGIC construye un directorio rejilla dividiento cada dominio en un conjunto de rangos acorde con los valores previamente obtenidos. Cada celda del directorio se corresponde con un fragmento de la relación que, en la última fase del proceso, es asignada a los procesadores pertinentes.

Aunque los resultados de rendimiento de la técnica MAGIC mostrados en [27] presentan valores muy aceptables, fundamentalmente en comparación a técnicas más usuales de particionamiento unidimensional o lineal (por un solo atributo), la carga de trabajo a que se somete el sistema para su evaluación sufre de tres serias debilidades:

- Las consultas emitidas involucran en sus predicados a un único atributo, lo que impide estudiar la característica de multidimensionalidad del método de particionamiento en la ejecución de una consulta aislada.
- Los predicados de las consultas referencian con exclusividad atributos que poseen valores únicos. Esta particularidad supone un obstáculo para la observación del método ante la presencia de sesgo en los datos.
- Se asume distribución uniforme a las tuplas de la relación. Esta consideración propone un escenario de medición que se encuentra lejos de situaciones realistas más propias de los sistemas de bases de datos al uso.

A los inconvenientes mostrados por los distintos métodos descritos, hay que unir la problemática permanentemente presente en los esquemas de particionamiento basados en la idea de los ficheros rejilla: la falta de equilibrio en la construcción de los rangos ante la

presencia de sesgo en los datos de entrada. Para entender con claridad este riesgo de desequilibrio, basta considerar que estos métodos establecen un número óptimo de particiones (rangos) en cada dimensión. Si los datos vienen sesgados es probable que muchas tuplas se concentren en pocos rangos, dejando muchas de las particiones sin contenido y haciendo entonces inútil la asignación de las celdas del directorio a los distintos nodos de procesamiento.

Los inconvenientes puestos de manifiesto por los mecanismos existentes de particionamiento multidimensional, nos empujan a definir un modelo de particionamiento que, en general, se ajuste a las siguientes exigencias:

- Particionamiento del espacio de tuplas de una relación en base a múltiples atributos.
- Capacidad para soportar operaciones de selección sobre los tres tipos de búsqueda básicos: coincidencia exacta, coincidencia parcial y búsqueda por rangos.
- Tolerancia a la presencia de sesgo en los datos, así como a la correlación existente entre varios atributos de la relación.
- Equilibrio en la asignación de las particiones entre los diferentes nodos, favoreciendo los criterios de escalabilidad y aceleración lineal del sistema multicomputador.
- Utilidad ante consultas en cuyo predicado aparezca cualquier número de atributos de particionamiento.
- Aplicabilidad del método ante situaciones en las que se posea un conocimiento previo de la carga de trabajo del sistema, así como otras en las que exista un desconocimiento de tal información.

En el siguiente apartado describimos un modelo general de particionamiento multidimensional basado en la estructura de árbol Q que, aunque por lógica comparte algunas similaridades con los modelos previamente comentados, acapara una serie de características propias que, a diferencia de otros trabajos previos sobre este particular, hacen viable la consecución de los objetivos previamente definidos.

4 Estrategias de particionamiento basadas en el árbol Q

En el presente apartado describimos dos estrategias generales de particionamiento multidimensional basadas en el árbol Q:

- Basada en el conocimiento previo de la carga de trabajo, la cual será referida en el resto del capítulo como la *estrategia BCCT* (Basada en el Conocimiento de la Carga de Trabajo).
- Basada en la existencia de un índice en árbol Q construido a priori, sin manejar información sobre la carga de trabajo. Ésta se denominará como la *estrategia BEPI* (Basada en la Existencia Previa del Indice).

La estrategia BCCT utiliza el conocimiento previo de la carga de trabajo ("*workload*") y las capacidades de procesamiento del sistema con el fin de construir un árbol Q "a medida", el cual será usado a modo de directorio de particionamiento y cuyos contenedores serán finalmente asignados a nodos de procesamiento durante la fase final del proceso de particionamiento. Este árbol Q construido a medida lo utiliza posteriormente el procesador de consultas para definir el modo de llevar a cabo la ejecución de aquellos operadores de selección que incluyan al menos uno de los atributos referenciados en el árbol Q.

Por su parte, la estrategia BEPI adopta una fórmula distinta para particionar la relación. En este caso, el proceso parte de un índice en árbol Q ya existente y distribuye los contenedores entre los nodos de procesamiento siguiendo tres políticas diferentes, a saber, las políticas de distribución de *grano grueso*, *grano medio* y *grano fino*.

Las dos estrategias generales de particionamiento y sus diferentes alternativas de aplicación se describen en los siguientes subapartados.

4.1 La estrategia BCCT

Esta estrategia de particionamiento resulta útil cuando se posee un conocimiento previo de la distribución de accesos a una relación. En esta situación es posible beneficiarse de dicha información construyendo un árbol Q expresamente adaptado a los futuros accesos a que se verá sometido. Este árbol Q dirigirá la asignación de las diferentes celdas del espacio de búsqueda entre un conjunto designado de nodos de procesamiento.

Para establecer el modo de funcionamiento de esta técnica de particionamiento, consideremos una carga de trabajo consistente en una serie de consultas de selección Q_i. Cada consulta Q_i recupera y procesa $\{Q_i\}$ tuplas de la relación. Con el fin de establecer la necesidad de recursos para una consulta típica, calculamos el número medio de tuplas que satisfacen una consulta típica de dicha carga de trabajo. A este valor lo denotamos por $\{Q_{avg}\}$.

En función de los recursos invertidos en la ejecución de una consulta media, se puede

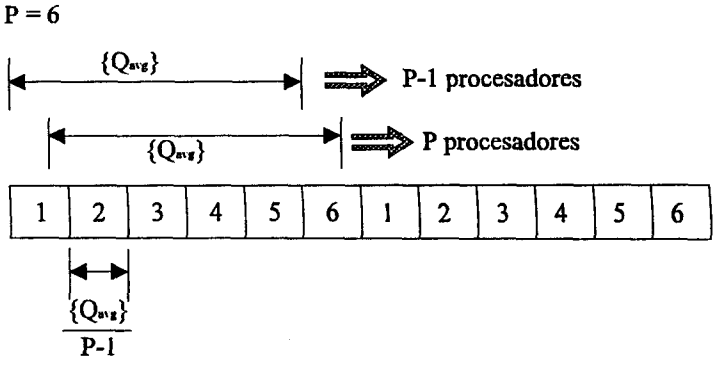


Figura 4-2. Asignación lineal de fragmentos de una relación a procesadores

determinar el número de procesadores P que deben participar en una consulta típica (este cálculo se establecerá en el momento en que se defina el modelo de ejecución de consultas, pues de otro modo sería imposible determinar los recursos invertidos y el coste de los mismos). Con el valor de P en nuestro poder, nuestro objetivo es hacer que todos los P procesadores participen en la selección de las $\{Q_{avg}\}$ tuplas. Fijado este objetivo, veamos cómo se podrían distribuir las tuplas de la relación para conseguir alcanzarlo.

Pongámonos en el caso unidimensional para establecer la mejor forma de distribuir las tuplas para que P procesadores intervengan en la selección de $\{Q_{avg}\}$ tuplas. En este caso, podríamos considerar el fichero ordenado en función del valor de un atributo. Cada celda de la partición contendría ahora un conjunto de tuplas cuyo valor sobre dicho atributo es mayor o igual a la de las tuplas de la celda anterior y menor o igual que el valor de dichas tuplas en la celda siguiente. Si cada celda la cargamos con un total de $\{Q_{avg}\}/(P\text{-}1)$ tuplas y asignamos P fragmentos adyacentes a otros tantos procesadores, podemos asegurar que al menos $P\text{-}1$ y a lo sumo P procesadores intervienen en el proceso de una consulta media. La figura 4-2 ilustra esta circunstancia para un total de $P = 6$ procesadores.

Esta forma de proceder, distribuyendo los fragmentos adyacentes entre el total de procesadores, es válida en el caso unidimensional, donde la respuesta a una consulta cuyo predicado involucra un único atributo, está compuesta por **todas** las tuplas en cada fragmento implicado en la solución.

Bajo una perspectiva multidimensional, las cosas suceden de forma diferente. En este caso, el espacio de búsqueda se encuentra fragmentado en base a múltiples atributos y las consultas involucran a uno o varios atributos de particionamiento. Esto hace que, desde el punto de vista geométrico, la forma de la respuesta a una consulta no concuerde necesariamente con particiones completas del espacio, de modo que es posible que **no todas** las tuplas de cada fragmento implicado en una respuesta satisfagan el predicado de la consulta, tal y como revela la figura 4-3a.

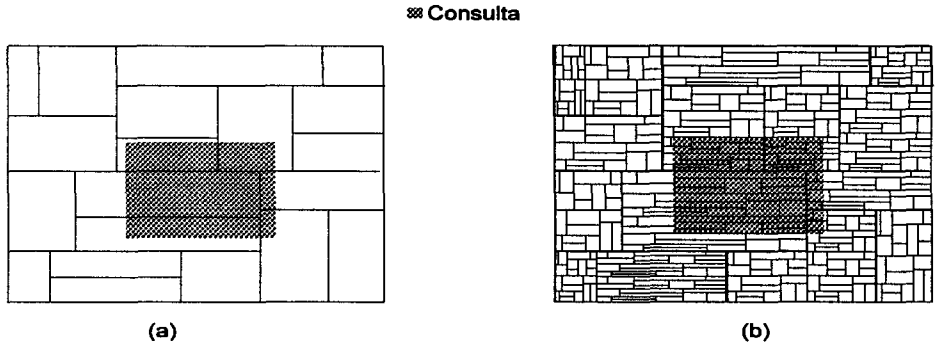


Figura 4-3. Particionamiento del espacio de búsqueda bidimensional. (a) Particionamiento grueso. (b) Particionamiento fino.

Tratando de generalizar el caso unidimensional, podríamos definir cada región del espacio con un tamaño para contener un total de $\{Q_{avg}\}/P$ tuplas. De este modo, podríamos asegurar que al menos P fragmentos adyacentes intervienen en la respuesta. Teniendo en cuenta que no todas las tuplas de un fragmento satisfacen necesariamente el predicado (figura 4-3a), el número máximo de fragmentos adyacentes que intervienen en la respuesta resulta difícil de determinar.

Sin embargo, tal y como muestra la figura 4-3b, considerando un particionamiento fino del espacio, los fragmentos implicados en la respuesta aportan un porcentaje mucho mayor de tuplas satisfaciendo el predicado. Como es obvio, mientras mayor sea este porcentaje, más se aproxima a P el número máximo de fragmentos participantes en la respuesta.

Considerando de modo general que un alto porcentaje de tuplas en cada contenedor involucrado en la respuesta a una consulta satisfará el predicado de la consulta, podemos en cierto sentido generalizar el caso unidimensional definiendo los contenedores con una capacidad tal que cada uno de ellos almacene aproximadamente $\{Q_{avg}\}/P$ tuplas.

Basada en esta idea, la estrategia BCCT distribuye de modo equilibrado los contenedores "adyacentes" entre los P procesadores previamente definidos. Esta distribución se realiza definiendo un tamaño de región tal que cada región gobierne aproximadamente un total de *Nº_total_de_contenedores* / P contenedores y repartiendo el contenido de cada región entre los P procesadores.

4.2 La estrategia BEPI

Bajo este nombre de estrategia, hemos considerado tres posibles alternativas para distribuir los contenedores de datos entre el conjunto de procesadores determinado. Las tres alternativas que proponemos se denominan de grano fino, grano medio y grano grueso, dependiendo del nivel en el que nos fijemos en el árbol y del modo de distribuir los contenedores.

El enfoque de **grano fino** "divisa" el espacio de búsqueda (la relación) como un conjunto de contenedores independientes. Esta alternativa utiliza en exclusividad el nivel hoja del árbol Q y distribuye los contenedores de forma equilibrada entre el conjunto predefinido de procesadores que deberían participar en la ejecución de una consulta sobre la relación. La distribución se lleva a cabo asignando de forma aleatoria el mismo número de contenedores a cada procesador en el conjunto. Con esta clase de reparto, intentamos equilibrar la carga de procesadores destruyendo cualquier posible patrón de acceso a los datos. En particular si existen zonas calientes del espacio de búsqueda, es decir, zonas de acceso frecuente por las consultas emitidas sobre la relación, este enfoque pretende dispersar en lo posible dichas zonas calientes entre distintos procesadores. La figura 4-4a muestra aporta una visión gráfica de este enfoque.

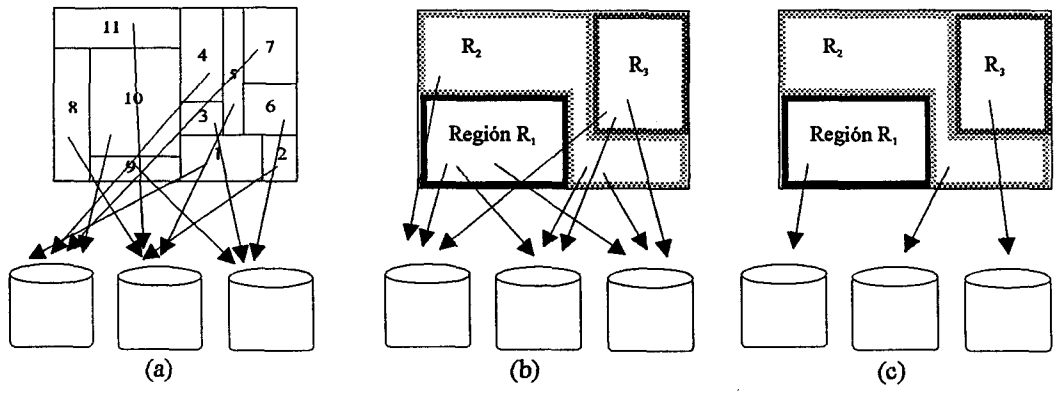


Figura 4-4. Filosofía de distribución de las estrategias BEPI. (a) Grano fino. (b) Grano medio. (c) Grano grueso.

En la distribución de **grano medio**, el espacio de búsqueda se percibe como un conjunto de regiones, cada una de las cuales gobierna un número parecido de contenedores. En este enfoque, el algoritmo de distribución usa los dos niveles más profundos del árbol Q, tomando separadamente cada región y asignando el mismo número de contenedores a cada procesador en el conjunto. Este método de distribución asegura, por ejemplo, la participación de una serie de procesadores distintos si la consulta se localiza sobre regiones específicas. En la figura 4-4b se muestra la forma en que se lleva a cabo la distribución de grano medio.

Por último, el uso de una distribución de **grano grueso** conduce a considerar el espacio como una mera partición en regiones disjuntas, sin entrar a considerar, salvo casos excepcionales de desequilibrio flagrante, la división en contenedores de cada región. En esta alternativa, el algoritmo de distribución recorre el nivel padre de los contenedores (es decir, el nivel de regiones) asignando de forma equilibrada regiones completas a procesadores. Ciertamente este reparto equilibrado dependerá en cierta medida del número de regiones, el tamaño de las mismas y el número de procesadores entre los que se decida realizar la distribución. En el caso en que la distribución sea inadecuadamente desequilibrada, debemos trocear algunas regiones para formar regiones más pequeñas y conseguir así el deseado equilibrio de carga entre procesadores. La distribución de grano grueso asegura, por ejemplo, la participación de varios procesadores en respuesta a consultas cuyo predicado viene descrito en términos de un sólo atributo. En la figura 4-4c aparece la forma de realizar la distribución en este enfoque.

Las estrategias de particionamiento previamente descritas han sido evaluadas en el marco de una sencilla máquina multicomputador y bajo una batería específica de pruebas ("*workload*"), construida al efecto de medir el rendimiento de aquellas propiedades que hemos considerado relevantes. Si bien los resultados experimentales se confinan al capítulo siguiente, el modelo de ejecución de consultas utilizado en el desarrollo de nuestros experimentos se define en el siguiente apartado.

5 El modelo de ejecución de consultas

Dada la imposibilidad de evaluar nuestras propuestas de particionamiento en una máquina paralela real en la que simular el comportamiento de nuestra base de datos QUATRO, el modelo de ejecución de consultas que ahora presentamos es un modelo teórico construido con el fin exclusivo de extraer resultados que nos permitan comparar las soluciones propuestas entre sí y con modelos de particionamiento clásicos. Tales comparaciones se efectúan básicamente en torno a cuatro parámetros de importancia demostrada en sistemas paralelos de memoria distribuida: tiempos de ejecución, equilibrio de carga entre procesadores, grado de aceleración ("*speedup*") y grado de escalabilidad ("*scaleup*").

De acuerdo a lo dicho, el modelo analítico de ejecución de consultas es relativamente sencillo. Cuando una consulta llega al sistema, el procesador de inicio/control/destino, utilizando el árbol Q y otra información de directorio, decide qué procesadores pueden contener tuplas relevantes que satisfagan el predicado de la misma. En este momento, la consulta se divide en tantas tareas como procesadores involucrados.

Por otra parte, cada tarea consume tiempo de CPU y realiza accesos físicos de E/S en cada procesador donde se ejecuta. Cuando una tarea finaliza su ejecución, espera en un bucle de sincronización hasta que finalicen sus tareas "gemelas" y la consulta pueda ser definitivamente cometida.

En lo referente a la comunicación, la ejecución sincronizada de la consulta procede como sigue:

1 El nodo de control emite un mensaje a cada nodo de procesamiento donde las tareas van a ejecutarse.

2 Cada nodo de ejecución envía un mensaje al nodo de control indicando que el procesamiento de la tarea se ha realizado y está listo para ser cometido.

3 Una vez que el nodo de control recibe el mensaje de "listo", enviado por todas las tareas, un mensaje de "cometer" ("*commit*") es dirigido a cada nodo de procesamiento con objeto de finalizar definitivamente la consulta.

4 Después de esto el resultado de la consulta puede ser enviado por los nodos de procesamiento al nodo de destino.

Considerando la red de interconexión como un medio de acceso común, tal como un bus, todas las tareas no pueden intercambiar datos en paralelo, de modo que el retraso provocado por la red en el intercambio de mensajes y envío de los resultados puede ser extraído del tiempo de dedicación de las tareas y considerado como tiempo invertido por el nodo de control/destino.

De acuerdo a estas consideraciones, el tiempo de respuesta de la consulta se define como la suma del tiempo de procesamiento T_{proc} y el tiempo de retraso inducido por la red T_{red}, donde T_{proc} es una función de los tiempos individuales de las tareas T_i, y viene definido por:

$$T_{proc} = \max(T_i)$$

y T_{red} es el retardo incurrido por la red en enviar los mensajes de comienzo y finalización de tareas, además del envío de la respuesta al procesador de destino, esto es:

$$T_{red} = \frac{\{Q_{avg}\}}{\omega_{comm}} + 3P\frac{I_{comm}}{\mu}$$

El primer sumando de esta fórmula computa el coste de envío de la respuesta y el segundo sumando el coste de flujo de mensajes para controlar el inicio y finalización de la tareas que componen la consulta.

Además de lo anterior, el tiempo de respuesta de una consulta paralela ejecutada por *P* procesadores soporta un coste ("*overhead*"), denominado coste de participación *CP*, incurrido por cada procesador adicional que participa en la ejecución de la consulta. Este coste viene dado principalmente en forma de mensajes adicionales para controlar la ejecución de la consulta y está estrechamente relacionado con el coste de comenzar una tarea sobre procesadores adicionales. El valor de *CP* es difícil de estimar a menos que tengamos un diseño completo de la máquina de base de datos donde la consulta será ejecutada. En nuestros experimentos, el valor de *CP* no se considera de forma explícita, salvo en la estrategia BCCT, en la cual este valor es un parámetro crítico para decidir el número óptimo de procesadores que deberían participar en la ejecución de la consulta.

El tiempo de tarea T_i para procesar una parte de la consulta Q en el procesador *i* conteniendo $\{Q_i\}$ tuplas que satisfacen el predicado de la consulta, viene definido por la fórmula

$$T_i = T_{cpu} + T_{io}$$

donde T_{cpu} es el tiempo de CPU, y viene dado por

$$T_{cpu} = \{Q_i\}\frac{I_t}{\mu}$$

y T_{io} es el tiempo invertido en los accesos físicos de E/S, el cual se define por

$$T_{io} = t_{acceso}(\{Q_i\})$$

Como puede apreciarse, este último término se define por medio de una función t_{acceso} cuyo argumento es el número de tuplas que finalmente recupera dicho nodo de procesamiento. Sin embargo, el tiempo de acceso a $\{Q_i\}$ tuplas depende del número de páginas de datos en las que residen dichas tuplas.

Con objeto de simplificar nuestro modelo, nosotros suponemos que la recuperación de una página de datos (contenedor) supone un tiempo de desplazamiento de las cabezas de lectura/escritura ("*seek*"), un retardo rotacional y por supuesto un tiempo de transferencia, y esto para cada acceso a un contenedor individual. Si consideramos cada contenedor del tamaño de una página física, este tiempo dependerá del número de contenedores en que dichas tuplas se localizan. De otro modo, el tiempo de acceso a un contenedor supone un tiempo de desplazamiento más una latencia rotacional para cada contenedor y un tiempo de transferencia del conjunto de bloques que ocupa un contenedor.

De acuerdo a estas hipótesis, el tiempo de acceso puede definirse por medio de la expresión

$$t_{acceso}(\|X\|) = \|X\|\delta_{io} + \frac{|X|\beta}{\omega_{io}}$$

donde ||X|| representa el número de contenedores que almacenan el item X, y |X| el número de páginas físicas en X.

Para poder comprender en toda su amplitud las diferencias de tiempo existentes entre los modelos de particionamiento BCCT y BEPI, es necesario describir con claridad las expresiones utilizadas para calcular los parámetros de configuración del árbol Q construido a medida en la aplicación de la estrategia BCCT.

En la implementación de la estrategia BCCT, hemos calculado el número óptimo de procesadores que deberían participar en la ejecución de una consulta típica Q_{avg}. Asumiendo de entrada equilibrio de carga entre procesadores, si *P* procesadores participan en la respuesta a esta consulta, cada nodo debería procesar $\{Q_{avg}\}/P$ tuplas, de modo que deberíamos pensar en el tiempo de respuesta *RT* de una consulta Q_{avg} como una función de *P*, definida como sigue:

$$RT(P) = \frac{\{Q_{avg}\}}{P}\frac{I_t}{\mu} + t_{acceso}\left(\frac{|Q_{avg}|}{P}\right) + T_{red}(P) + CP(P)$$

donde *CP* es el costo de participación de procesadores adicionales. Nosotros consideramos este valor como una función lineal de *P*, de modo que sustituyendo los símbolos correspondientes, tenemos

$$RT(P) = \frac{\{Q_{avg}\}}{P}\frac{I_t}{\mu} + \frac{|Q_{avg}|}{P}\left(\delta_{io} + \frac{\beta}{\omega_{io}}\right) + \frac{\{Q_{avg}\}}{\omega_{comm}} + 3P\frac{I_{comm}}{\mu} + P*CP$$

Resolviendo ahora la ecuación *RT'(P)* = 0, podemos obtener el número óptimo de procesadores *P* que deberían participar en la respuesta de Q_{avg}:

$$P = \sqrt{\frac{\{Q_{avg}\frac{I_t}{\mu} + |Q_{avg}|\left(\delta_{io} + \frac{\beta}{\omega_{io}}\right)}{3\frac{I_{comm}}{\mu} + CP}}$$

Tabla 4-3. Lista de términos y parámetros de simulación

μ	Velocidad del procesador (2 MIPS)
I_t	Coste de procesamiento de una tupla (1000 instrucciones)
I_{comm}	Coste de intercambiar mensajes por la red (1000 instrucciones)
ω_{io}	Ratio de transferencia de E/S (1.2 Mb/s)
ω_{comm}	Ancho de banda efectivo de la red (2 Mb/s)
δ_{io}	Latencia media de E/S (18.3 ms)
t_{acceso}	Tiempo de acceso a disco = $\|\|Q\|\|*\delta_{io} + (\|Q\|*\beta)/\omega_{io}$
β	Tamaño de bloque (2 Kb)
$\{X\}$	Número de tuplas en el ítem X
$\|X\|$	Número de bloques en el ítem X
$\|\|X\|\|$	Número de contenedores en el ítem X

El valor obtenido de *P* será utilizado en la forma en que se describe en 4.3 para definir el tamaño de los contenedores del árbol Q en la estrategia BCCT.

Como ya hemos adelantado, aunque el modelo de ejecución de consultas es relativamente simple, realmente permite capturar el comportamiento de las diferentes estrategias de particionamiento sin interferencias de ningún efecto lateral causado por agentes externos al propio modelo de particionamiento. Por esta razón, los tiempos que se obtienen como resultado de las distintas pruebas experimentales a las que hemos sometido nuestras propuestas no son representativos fuera del ámbito descrito, en su lugar tales valores son tratados como elementos básicos de comparación entre las diferentes estrategias de particionamiento.

Para finalizar este apartado, la tabla 4-3 incluye una lista de términos utilizados en las fórmulas previas, así como los parámetros más importantes de la simulación efectuada del modelo de ejecución de consultas.

6 Paralelización del árbol Q

Aunque los trabajos efectuados sobre la explotación del árbol Q en sistemas paralelos se centran en la propuesta de estrategias de particionamiento multidimensional de datos, nos parece interesante exponer aquí, a modo de avance, las lineas generales que estamos considerando en el establecimiento de mecanismos de distribución del índice.

Existen dos opciones básicas que pueden ser adoptadas en el proceso de distribución y posterior uso paralelo del índice en árbol Q: distribución estática o construcción dinámica distribuida del índice.

La distribución estática consiste en construir de forma centralizada, esto es en un sólo nodo, el índice en árbol Q y, una vez finalizado, realizar un particionamiento equilibrado de los Q-nodos entre los distintos nodos de procesamiento de la máquina. La forma natural de efectuar tal distribución consiste en almacenar en cada nodo de procesamiento (considerando el nodo con capacidad de almacenamiento secundario) una región del árbol Q, repitiendo la operación para los niveles superiores del árbol Q.

Por su parte, la construcción dinámica distribuida del índice introduce una problemática más atractiva a la explotación paralela del índice. En este caso, la idea básica estriba en comenzar la construcción del árbol en un nodo específico de la máquina multicomputador, de modo que, una vez superado el nivel de carga de dicho nodo, éste tenga la capacidad de delegar en otro nodo distinto el crecimiento y gestión de una parte del árbol Q, repartiendo así las tareas de mantenimiento del índice entre varios nodos de la máquina y favoreciendo, adicionalmente, la participación de varios nodos en la ejecución de una consulta.

Considerando el amplio espectro de posibilidades que ofrece la perspectiva dinámica del problema, nuestros esfuerzos se centran actualmente en plantear diversas estrategias de crecimiento distribuido del árbol Q. Destacamos a continuación aquellas que parecen aportar a priori soluciones viables al problema en cuestión y sobre las cuales encaminamos actualmente nuestros estudios.

6.1 Uso de hiperpáginas

El concepto de hiperpágina o multicubo ya fue originalmente introducido por Lomet en [43]. En nuestro caso, esta estrategia de distribución dinámica consiste en considerar *grandes* páginas con capacidad para almacenar un conjunto predefinido de contenedores de datos. La capacidad de esta *hiperpágina* vendrá impuesta por la carga máxima asignada a un procesador (con disco) concreto. Los contenedores de una hiperpágina se mantienen organizados según un árbol Q.
En el momento del desborde de una hiperpágina, ésta se divide, apareciendo una nueva hiperpágina que se asigna a un procesador distinto. El árbol Q se fracciona entonces en dos partes, cada una de las cuales va a parar a un procesador distinto. Ya que los contenedores en una hiperpágina no siguen ninguna secuencia lógica, la forma de dividir una hiperpágina difiere necesariamente de aquella que se utilizaba en la división de un contenedor. En este caso la división se efectúa basándonos en la información de los niveles superiores del índice repartiendo, por ejemplo, un número similar de regiones entre los nodos de procesamiento implicados en la división.

Una vez realizada la división, podríamos decir que el nodo conteniendo la raíz del árbol Q delega en otro el crecimiento del mismo por alguna de sus ramas, de modo que cada solicitud a esa parte del árbol es, a partir de la división, servida por un nuevo nodo de procesamiento. En esta estrategia cada nodo de procesamiento indexa los datos que allí se almacenan.

Aunque este enfoque posee el inconveniente del movimiento de un volumen considerable entre los nodos de procesamiento, ofrece una perspectiva homogenea de la localización distribuida del índice que favorece el concepto de localidad de los datos, es decir, aquellos datos con propiedades espaciales similares se localizan próximos físicamente.

6.2 Asignación independiente de contenedores y regiones

Contrariamente a la anterior, en esta perspectiva de distribución no existe relación directa entre la localización física de los Q-nodos índice y el lugar en que se almacenan los contenedores. En el caso que nos ocupa, los contenedores se van asignando a los distintos procesadores con independencia del lugar en que se localiza la región que los gobierna. En el momento de la división de una región, las regiones resultantes se distribuyen de modo similar entre distintos procesadores.

Esta forma de distribución, al contrario que la anterior, rompe cualquier conexión física entre Q-nodos con relación de descendencia directa en el árbol Q. Así por ejemplo, dos contenedores y la región que los gobierna pueden estar localizados en tres nodos de procesamiento distintos de la máquina multicomputador, lo que puede, en algún sentido, ralentizar las respuestas a consultas de tipo rango en las cuales dicho rango se concentre en una pequeña zona específica del espacio de búsqueda. Además de esto, debido a la dispersión de los distintos Q-nodos, su situación física dentro de la máquina debe ir necesariamente acompañando a la identificación del mismo para su localización efectiva en el recorrido del árbol. Esta información extra supone un gasto adicional de espacio en el interior de los Q-nodos que hace disminuir el grado de utilización de espacio y por tanto la descendencia (*"fan-out"*) del árbol.

Por contra, este enfoque tiene la gran ventaja de no requerir un movimiento masivo de datos entre los nodos de procesamiento de la máquina, evitando el retraso debido al consumo de los recursos de red durante la fase de crecimiento del árbol.

El lector interesado en obtener detalles más específicos sobre el estado actual del desarrollo de este proyecto de distribución dinámica del árbol Q, puede remitirse a [48].

7 Conclusiones

El estudio de la problemática del particionamiento de relaciones y, como consecuencia, la explotación del árbol Q en sistemas paralelos, han constituido el argumento central del presente capítulo.

A diferencia de los mecanismos habituales de particionamiento proporcionados por típicos SGBDP, donde la relación se fragmenta sobre la base de un único atributo, el método que aquí presentamos, basado en la explotación del árbol Q, es capaz de particionar las relaciones usando múltiples atributos.

La descripción de otros métodos de particionamiento multidimensional analizados en este capítulo, nos permiten identificar una serie de propiedades exclusivas de nuestro mecanismo de particionamiento, tales como la capacidad de soportar con eficiencia consultas de tipo rango, así como de coincidencia parcial y exacta, la tolerancia ante la presencia de sesgo en los datos de la relación, o la utilidad de nuestras propuestas ante consultas en cuyo predicado aparezca un diferente número de atributos de particionamiento.

Las labores de investigación llevadas a cabo sobre la estructura de árbol Q y su explotación como soporte de las distintas propuestas de particionamiento multidimensional, quedarían incompletas sin una valoración experimental de los mismos. La imposibilidad de contar con un sistema multiprocesador real donde ejecutar las baterías de pruebas diseñadas para la evaluación de los métodos propuestos, nos ha llevado a definir un modelo simulado de ejecución de consultas que nos posibilitará aproximar el rendimiento de las estrategias de particionamiento diseñadas. El modelo de ejecución que hemos presentado, aunque sencillo, se ajusta fielmente a otros utilizados para medir el rendimiento de operaciones relacionales en sistemas paralelos [12, 35, 41, 61].

Los resultados obtenidos a partir de tales trabajos de evaluación nos han permitido, primeramente, corroborar el correcto planteamiento de la estructura de árbol Q y, finalmente, emitir un juicio razonado acerca del comportamiento de los mecanismos de particionamiento basados en esta estructura.

Por otra parte, los estudios actuales al respecto de la explotación del árbol Q en SGBDP, no se detienen en las propuestas de mecanismos de particionamiento de relaciones, sino que se extienden a la propia distribución del índice en tales sistemas. En este sentido, un método de distribución del árbol Q, conocido como método DQG ("*Dynamic Q-Growth*"), está siendo actualmente desarrollado como extensión a los trabajos aquí presentados. Las lineas generales de actuación al respecto de la distribución del índice en árbol Q han sido presentadas a modo de avance en este capítulo.

El trabajo experimental, cuyo desarrollo ha constituido una parte esencial en el planteamiento de la tesis que presentamos, se describe con detalle en el siguiente capítulo.

Capítulo 5

Resultados experimentales

En este capítulo aportamos los diferentes resultados de rendimiento, tanto de la estructura de árbol Q, como de su aplicación al particionamiento de ficheros en bases de datos paralelas. Los resultados de evaluación que aquí mostramos son producto del trabajo de implementación llevado a cabo en distintas fases de la elaboración de esta tesis. Si bién la implementación de la estructura de árbol Q fue perfectamente abordable desde un principio con los medios disponibles, su aplicación como método de particionamiento en máquinas paralelas tuvo que ser simulada a través de un modelo sencillo de ejecución de consultas en una máquina multicomputador. Aunque muchos detalles concernientes a la implementación han sido intencionadamente omitidos, en el presente capítulo se destacan los aspectos genéricos del trabajo de implementación del árbol Q y los diferentes resultados, relativos al rendimiento de las diferentes propuestas, obtenidos como consecuencia de nuestra labor experimental.

Con objeto de presentar tales resultados, el capítulo se organiza en tres grandes apartados. El apartado 1 aporta una descripción global de las estructuras de datos utilizadas en la implementación del árbol Q, así como una visión general de los distintos módulos que componen el programa de aplicación que lleva a cabo la construcción del árbol Q.

El apartado 2 expone los resultados de rendimiento respecto a la estructura de árbol Q en sí misma. Para presentar ordenadamente estos resultados, este apartado se divide en tres partes. El subapartado 2.1 presenta las tres estrategias de construcción consideradas para el árbol Q,

el subapartado 2.2 describe las baterías de pruebas utilizadas en la evaluación de la estructura y, para finalizar este apartado, el punto 2.3 aporta diferentes figuras de rendimiento del árbol Q.

Por su parte, el apartado 3 describe los experimentos llevados a cabo sobre la utilización del árbol Q en sistemas paralelos y destaca los aspectos más relevantes acerca de dichos experimentos. Así, el subapartado 3.1 describe la batería de pruebas utilizada para realizar la evaluación de las diferentes alternativas de particionamiento propuestas, mientras que en el subapartado 3.2 se exhiben y comentan las figuras de rendimiento obtenidas como resultado de las pruebas experimentales.

1 Implementación del árbol Q

Tal y como hemos comentado con anterioridad, la fase experimental de la tesis que presentamos cobra una importancia definitiva en el grueso del trabajo. A pesar de la profundidad con la que se plasman los algoritmos de construcción del árbol Q y la rigurosidad con la que se formalizan sus propiedades, esta labor de definición y desarrollo de las ideas que han conducido al nacimiento de esta nueva estructura de indexación, podrían tener una validez meramente simbólica sin el refrendo de unos resultados obtenidos a partir de su implementación. Conviene por ello indicar que, independientemente del pequeño porcentaje de páginas en las que se tratan aspectos de la implementación, la labor efectuada en este sentido ha supuesto una inversión de tiempo tan importante como las tareas de investigación básica.

Aunque no es nuestra intención narrar aquí la forma en que hemos llevado a cabo la implementación del árbol Q, sí nos parece interesante resaltar aquellos aspectos genéricos más relevantes del trabajo de implementación desarrollado.

Los programas de aplicación que definen y manipulan la estructura de árbol Q han sido escritos en lenguaje C y el conjunto de módulos que componen la aplicación principal encargada de insertar datos en el árbol Q se extiende a un total de 17.200 lineas de código fuente.

Desde un punto de vista físico, el árbol Q se almacena en dos ficheros: el fichero de datos, donde se incluyen los contenedores, y el fichero índice, conteniendo los Q-nodos de la zona de índices del árbol, es decir regiones y espacios.

Ambos ficheros se organizan como un conjunto de páginas todas del mismo tamaño. Una página cabecera almacena distinta información de control y administrativa, necesaria para la gestión de ambos ficheros, tales como número actual de páginas ocupadas, identificación de la página raíz del árbol Q, etc.

Respecto a los contenedores, la estructura de la página es sencilla y contiene una pequeña cabecera con información administrativa, seguida por un array cuyos elementos albergan los registros de datos (tuplas de la relación). En la primera versión del árbol Q y con objeto de simplificar su manejo, cada tupla se organiza internamente utilizando una técnica de representación con campos de longitud fija [30]. De este modo, cada página tiene espacio para almacenar el mismo número de tuplas.

La organización interna de las páginas del índice es algo más compleja. En este caso, cada página se corresponde con un Q-nodo de la zona de índices y almacena su árbol k-d local. Como en las páginas de datos, cada página incluye una cabecera con información administrativa.

El árbol k-d local se prepresenta por medio de un array cuyo primer elemento contiene la información del nodo local raíz. Así pues, la estructura de una página viene definida como:

```
struct pagina_indice {
    INFO_CAB            cabecera;
    INFO_NODO_LOCAL   arbol-kd-local [Max_num_entradas];
    }
```

Cada elemento del array que representa el árbol es un nodo local que puede ser interno u hoja, dependiendo de lo cual, así almacena la información pertinente. De este modo, el tipo INFO_NODO_LOCAL se define como

```
typedef union {
    INFO_NODO_LOCAL_INTERNO      nodo_interno;
    INFO_NODO_LOCAL_HOJA         nodo_hoja;
    } INFO_NODO_LOCAL;
```

Por una parte, el nodo interno almacena el atributo discriminador, el valor del mismo, banderas de subido e inclusión, posible valor de clave primaria y dos apuntadores a los nodos hijos, cuyo contenido es el subíndice del array arbol_kd_local[] correspondiente a su nodo hijo. Por su parte, un nodo hoja almacena un apuntador al Q-nodo hijo junto con la indicación de si tal Q-nodo es un contenedor (acceso a un fichero diferente) o es una página del índice (acceso al mismo fichero).

La implementación del árbol Q se ha dispuesto en una serie de módulos conteniendo funciones que proporcionan diferentes servicios. Aportamos a continuación una breve descripción de cada uno de estos módulos:

- **Módulo de E/S**. Proporciona servicios básicos de acceso a las páginas de los diferentes ficheros. Adicionalmente contiene una serie de funciones de utilidad relativas a la gestión de entrada/salida.

- **Módulo de gestión de páginas**. Es el encargado de operar con la información interna a las páginas, tanto de datos como de índice.
- **Módulo de división de contenedores**. Se encarga de llevar a cabo la división de un contenedor cuando la página destino no tiene sitio libre para almacenar una nueva tupla.
- **Módulo de división de páginas del índice**. Es el más complejo de todos, proporcionando los servicios necesarios para dividir una página del índice cuando la información procedente del nivel inferior no cabe en la página destino. Este módulo implementa la inmensa mayoría de los algoritmos que, a un alto nivel de abstracción, fueron descritos en el capítulo 3.
- **Módulo de utilidades**. Es un módulo genérico conteniendo un conjunto de funciones de propósitos muy diversos, necesarias para realizar ciertos servicios solicitados por el resto de los módulos de la aplicación.

Las estructuras y módulos aquí descritos constituyen el soporte básico para la realización de las pruebas desarrolladas al objeto de evaluar el rendimiento, tanto del mecanismo de indexación en sí, como de las propuestas de particionamiento multidimensional basadas en esta estructura. Los resultados aludidos se presentan en los siguientes apartados.

2 Evaluación del árbol Q

2.1 Estrategias de construcción del árbol Q

En el capítulo 3 se describió con detalle la forma de construir, mantener y utilizar el índice en árbol Q. En concreto se vio cómo cuando un contenedor se desborda es necesario dividirlo. Para ello se elige una dimensión apropiada y un valor para dicha dimensión que permita repartir los datos entre el contenedor sobrecargado y un nuevo contenedor.

La definición original del árbol k-d, el cual -no olvidemos- es la estructura elegida para representar la división de contenedores dentro de una región, presupone una elección cíclica de discriminadores sobre los cuales se realiza la división de una hoja. Así, en un nivel concreto del árbol k-d, se utiliza una misma dimensión para llevar a cabo la división de cualquier contenedor situado en ese nivel. Normalmente, en el nivel i se utiliza la dimensión $(i \bmod k) + 1$, siendo k el número de dimensiones del dominio de búsqueda. Si bien esta estrategia de selección de la dimensión discriminante es aplicable a nuestra estructura, no es la única y desde luego no se trata de la mejor estrategia en todos los casos.

El problema de la elección del atributo o dimensión discriminante ha sido tratado con anterioridad en diferentes trabajos tales como en [9, 26]. Sin embargo, en el ámbito de árboles Q, la cuestión acerca de qué atributo seleccionar en cada división de un contenedor conlleva una serie de particularidades propiamente específicas de nuestro método de acceso. Estas particularidades pueden resumirse en los siguientes puntos:

- La posible aparición de la clave primaria como parte del discriminador de contenedores hace viable la utilización de cualquier dimensión como atributo discriminante. Sin embargo, a medida que aumente la presencia de valores de clave primaria como parte del discriminante, disminuye el rendimiento de las búsquedas en el árbol Q.
- Debido a la imposición de un nivel de ocupación mínimo para cada contenedor, sin la participación de clave primaria, algunos atributos podrían no ser susceptibles de utilizarse como discriminantes en el momento de la división de ciertos contenedores, pues es posible que no exista ningún valor para un atributo concreto tal que un conjunto **limitado** de tuplas en el contenedor tengan sobre dicho atributo valores menores o iguales que aquél.

El primero de los puntos anteriores nos abre completamente el abanico de posibilidades de elección de la dimensión discriminante. Por contra, el segundo de ellos nos restringe en cierta medida el número de atributos que pueden ser utilizados a la hora de dividir un contenedor sobrecargado. Esto significa, en definitiva, que sería conveniente mantener el criterio según el cual el número de dimensiones elegibles viene impuesto por la restricción de carga mínima de los contenedores, en tanto y cuanto imposiciones de otra índole nos permitan elegir como discriminante alguna de las dimensiones en ese conjunto restringido. En cualquier caso, siempre que dichas imposiciones externas entren en conflicto con el criterio de limitación de dimensiones discriminantes, el concurso de la clave primaria nos va a permitir seleccionar el atributo más aconsejable, cualquiera que sea éste.

Las imposiciones referidas en el párrafo anterior son principalmente de naturaleza estructural y semántica, teniendo en cuenta el tipo de índices con el que estamos tratando. Estas imposiciones pueden resumirse en las siguientes:

- Siendo k el número de dimensiones sobre las cuales se van a efectuar habitalmente las búsquedas en el índice, es necesario que todas ellas hayan sido en algún momento seleccionadas como discriminantes en la división de un contenedor. Esto significa que, siendo k_i una dimensión concreta del dominio de búsqueda, el árbol k-d local a una región debería incluir, al menos en uno de sus nodos locales, la dimensión k_i como atributo discriminante, ya sea en solitario o acompañada por un valor de clave primaria.

- Siempre y cuando otorguemos la misma importancia a todas las claves de búsqueda en nuestro dominio, la utilización de dimensiones discrimantes debería llevarse a cabo bajo criterios de uniformidad, de modo que aproximadamente el mismo número de ocasiones aparezca cada dimensión k_i en los nodos locales a una región.
- En caso de poder estimar con anterioridad las frecuencias con la que los diferentes atributos aparecen en las búsquedas a efectuar sobre el árbol Q, un vector de pesos asociado a cada dimensión en función de su importancia, jugaría un papel importante a la hora de determinar qué atributo debe ser elegido en la división de un contenedor.
- Fijado, por último, un recorrido desde la raíz del árbol Q hasta un contenedor concreto, dos circunstancias serían deseables. Por una parte, las k dimensiones deberían aparecer a lo largo del recorrido. En segundo lugar, la frecuencia con la que aparece una dimensión k_i debería estar en consonancia con el peso que se otorga a dicha dimensión, según hemos establecido en los dos puntos anteriores.

Las anteriores consideraciones sientan las bases para el establecimiento de las diferentes estrategias de selección de discriminadores que hemos considerado en el desarrollo de la presente tesis.

Conviene reseñar que, si bien no son las únicas posibles, las estrategias aquí propuestas son las que hemos considerado en el transcurso de los análisis efectuados con objeto de medir el comportamiento del árbol Q. Un estudio analítico exhaustivo acerca de las diferentes alternativas que existen respecto a la elección de la dimensión discriminante, aunque inicialmente se enmarca fuera del ámbito de esta tesis, queda como propuesta adicional de trabajo futuro.

Tres son las alternativas consideradas de selección de la dimensión discriminante en el momento de la división de un contenedor. La primera y más básica es la estrategia de selección *por niveles*, la segunda que hemos denominado del *mejor atributo* y la última concocida como estrategia de selección *por pesos*. Describimos a continuación cada una de ellas.

2.1.1 Estrategia de selección por niveles

La estrategia de selección por niveles es la más simple de todas y se lleva a cabo seleccionando en cada nivel del árbol local a una región, una dimensión establecida de antemano. La forma de establecer, para un nivel i, el atributo discriminante a utilizar en la división del contenedor (hoja del árbol local) situado en dicho nivel, se determina registrando en cada contenedor resultante de la división, la dimensión discriminante utilizada en dicha división. Cuando alguno de los contenedores resultantes de la división se tenga que dividir de nuevo, se selecciona la siguente dimensión, es decir se utiliza la dimensión $(m \bmod k) + 1$, siendo m la dimensión que aparece registrada en el contenedor que se va dividir, es decir, aquella que se utilizó en la anterior división y k el número de dimensiones del espacio.

Con este método de selección nos aseguramos la participación de todas las dimensiones en los nodos locales a las regiones. Además, dado un camino desde la raíz del árbol Q hasta un contenedor, estamos seguro de que todos los atributos aparecen con una frecuencia similar en tal recorrido.

La mayor desventaja del método de selección por niveles radica en su falta de flexibilidad. Si una gran cantidad de tuplas del contenedor poseen el mismo valor sobre el atributo discriminante, la división que se consigue puede no garantizar el requisito de utilización mínima del contenedor, con lo que será preciso utilizar adicionalmente el valor de la clave primaria para efectuar la división.

2.1.2 Estrategia de selección del mejor atributo

En esta estrategia, la idea básica consiste en llevar una contabilidad del número de ocasiones en que cada dimensión ha sido utilizada como discriminante. Para ello, dentro de cada contenedor almacenamos un vector de k componentes (llamado vector de uso de discriminantes), cuyo valor para cada componente no es otro que el número de ocasiones en que la dimensión correspondiente ha sido utilizada en las divisiones previas. En el momento de la división de un contenedor seleccionamos, a partir del vector de uso de discriminantes v[], aquellas componentes j tales que

$$v[j] = Min\,(v[i]),\ i = 1..k \qquad [1]$$

De entre los atributos correspondientes a las dimensiones j obtenidas según la expresión [1], seleccionamos aquél que consiga la división más uniforme posible. Si ninguna de estas divisiones satisface el requisito de carga mínima, entonces alguno de los atributos para los que se satisface la expresión [1], junto con un valor adicional de clave primaria, se selecciona como atributo discriminante.

A diferencia del anterior, este método aporta una mayor flexibilidad, ya que la dimensión discriminante no se prefija de antemano. Con ello se consigue una distribución lo más uniforme posible en el momento de la división de un contenedor sin el coste adicional que el uso de la clave primaria conlleva en el proceso de búsqueda en el árbol Q. Además, con esta estrategia mantenemos, igual que en el caso de la selección por niveles, el criterio de uniformidad de uso de los distintos atributos discriminantes.

La principal desventaja de esta estrategia estriba en la necesidad de examinar con mayor detalle las posibles divisiones de la página con el concurso de varios atributos discriminantes distintos. Esto se traduce en un mayor tiempo de proceso del contenedor en el momento de la división y, en consecuencia, en un mayor coste de mantenimiento del árbol Q.

2.1.3 Estrategia de selección por pesos

La última de las estrategias de selección de la dimensión discriminante que analizamos consiste en elegir, para cada división, el atributo discriminante en función de la importancia (el peso) que otorguemos a los diferentes atributos del dominio de búsqueda.

La idea que subyace en la estrategia de selección por pesos es la de favorecer la aparición de determinadas dimensiones (aquellas más frecuentemente requeridas en las búsquedas), en perjuicio de aquellas que aparecen con poca frecuencia en los predicados de las consultas.

Para llevar a cabo la elección de una dimensión en el momento de la división de un contenedor, un vector de pesos indicando la frecuencia sugerida para cada dimensión constituye la base del cálculo que conduce a la elección. Un contador registra el número de divisiones globales realizadas sobre el dominio de búsqueda. Además de ello, un vector *uso*[], con k componentes, registra el número de ocasiones en que cada dimensión ha sido seleccionada como discriminante. En el evento de una nueva división se calcula, para cada dimensión i, la proporción que corresponde a su peso, esto es

$$prop[i] = n°\_divisiones * peso[i]$$

posteriormente la dimensión cuyo uso se encuentre más alejada del peso otorgado es la que se selecciona, esto es, el subíndice j correspondiente a

$$Máx\ (|prop[i] - uso[i]|),\ i = 1..k$$

En el caso de que varias dimensiones consigan este máximo, aquella que mejor división provoque, es decir, la que reparta más uniformemente las tuplas en el contenedor sobrecargado, será la dimensión elegida.

2.2 Baterías de pruebas

En la evaluación de nuestra estructura índice en árbol Q nos planteamos tres objetivos básicos:

1 Comprobar el comportamiento del árbol Q en diferente número de dimensiones, tanto con respecto a la utilización del espacio, como en relación al tamaño del índice.

2 Medir los efectos de la elección de la estrategia de división de contenedores respecto al rendimiento en las búsquedas.

3 Verificar el buen comportamiento del árbol Q frente a distribuciones realistas de los datos de entrada.

Tabla 5-1. Detalle de los atributos de la relación utilizada en la Batería 1.

Nombre	Dominio	Observaciones
Unico	0-(Maxtuplas -1)	aleatorio, único, 4 bytes
Dos	0-1	aleatorio, 2 bytes
Cuatro	0-3	aleatorio, 2 bytes
Diez	0-9	aleatorio, 2 bytes
Veinte	0-19	aleatorio, 2 bytes
UnoPorciento	0-99	(Unico mod 100), 2 bytes
DiezPorciento	0-9	(Unico mod 10), 2 bytes
VientePorciento	0-4	(Unico mod 5), 2 bytes
CincuentaPorciento	0-1	(Unico mod 2), 2 bytes
ParUnoPorciento	0, 2, 4, ..., 198	(UnoPorciento*2), 2 bytes
ImparUnoPorciento	1, 3, 5, ..., 199	(UnoPorciento*2 + 1), 2 bytes

El primero de los objetivos se establece ante la posible viabilidad del árbol Q como método de acceso a datos espaciales. Dado que la representación de objetos espaciales precisa de una cantidad mayor de información para almacenar sus fronteras, una estructura índice multidimensional demasiado sensible al incremento en el número de dimensiones representa un serio obstáculo en el tratamiento de tales objetos, ya que el índice puede crecer indiscriminadamente hasta alcanzar tamaños poco manejables. A pesar de que la aplicación del árbol Q en sistemas de tratamiento de datos espaciales es un compromiso de futuro, este objetivo nos permite comprobar su posible utilidad en este ámbito de aplicación.

Para alcanzar el objetivo 1, utilizamos una versión adaptada de la relación original de la batería de pruebas del "*Wisconsin Benchmark*" [29]. Los datos de entrada consistieron en un total de 100.000 tuplas de 24 bytes cada una. Todos los atributos son enteros de 2 bytes, salvo la clave primaria que ocupa 4 bytes. La tabla 5-1 ofrece una especificación más detallada de cada uno de los atributos esta relación. Para evaluar la adaptabilidad del árbol Q al incremento en el número de dimensiones, se construyeron árboles con tres tamaños de página diferentes de 1Kb, 2Kb y 4Kb. A esta batería de pruebas la denominamos Batería 1.

Los objetivos 2 y 3 se atacaron con una batería diferente, la cual nos permitiera un mayor control de la distribución de los datos de entrada. La Batería 2 consistió en una relación EMP (Unico, Ciudad, Edad, Salario, Dept, Categoria) que hace referencia a datos sobre empleados de una compañía cuya información se almacena en términos de los seis atributos mencionados. La distribución de los valores sobre dichos atributos es la siguiente:

- *Unico* toma valores únicos para cada empleado y representa la clave primaria de la relación.

Tabla 5-2. Detalles de la relación utilizada en la Batería 2.

Atributo	Dominio (Tamaño)	Tipo	Distribución	Observaciones
Unico	100.000	entero	secuencial	único, no consultado
Ciudad	10	cadena	1-35%, 1-25%, 8-40%	consultas de igualdad
Edad	25	entero	10-70%, 15-30%	consultas de igualdad y rango
Salario	100	entero	30-90%, 70-10%	consultas de rango
Dept	15	cadena	uniforme	consultas de igualdad
Categoría	5	entero	2-90%, 3,10%	consultas de igualdad y rango

- *Ciudad* informa sobre la ciudad donde el empleado presta sus servicios. Sus valores se obtienen de un dominio con diez valores distintos, distribuidos de modo que una ciudad aparece en el 35% de las tuplas, otra ciudad en el 25% de las tuplas, y el resto de valores del dominio se distribuyen uniformemente entre el resto de tuplas de EMP.
- *Edad* posee 25 valores distintos, de los cuales 10 de ellos se concentran en el 70% de las tuplas y el resto, uniformemente entre el 30% restante de las tuplas.
- *Salario* es un atributo con 100 valores posibles que sigue una distribución 90:30, es decir 30 de esos valores aparecen en el 90% de las tuplas.
- *Dept* referencia al departamento del empleado y toma 15 valores diferentes uniformemente distribuidos.
- Por último, el atributo *Categoría* toma sus valores de un dominio de 5 valores posibles, donde sólo dos de ellos aparecen en el 90% de las tuplas.

La tabla 5-2 resume las características básicas de la relación utilizada para las pruebas de la Batería 2. Detallamos a continuación el contenido de las pruebas incluídas en esta batería.

De acuerdo a estas distribuciones, se insertaron 100.000 tuplas en un árbol Q indexado por cinco dimensiones (todos los atributos salvo la clave primaria). Ya que en este caso no fue nuestra intención discriminar ninguno de los atributos clave secundaria en particular, sólo las dos primeras estrategias de selección de atributos fueron consideradas en nuestros experimentos[1].

Una vez insertadas las tuplas procedentes de la relación EMP en los correspondientes árboles Q (uno para cada estrategia de selección utilizada), se emitieron cinco tipos distintos de consultas, denominadas Q_1 a Q_5. El subíndice del tipo de consulta referencia el número de atributos involucrados en una consulta de ese tipo. Así, cada tipo de consulta Q_i, se construye

1 La estrategia de selección por pesos cobra relevancia durante la evaluación de las distintas estrategias de particionamiento multidimensional.

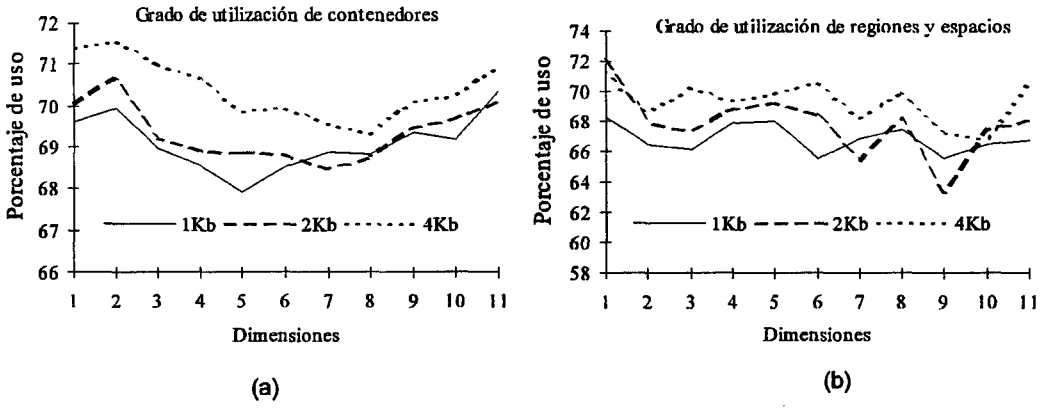


Figura 5-1. Grados de utilización de las páginas del árbol Q.

con predicados en los que aparecen *i* atributos conectados por el operador "*AND*". Por su parte, cada tipo de consulta se compone de un total de 500 instancias con sentido. Así por ejemplo, sobre el atributo *Ciudad* se contemplaron con exclusividad predicados de igualdad, mientras que sobre el atributo *Salario*, las consultas podían incluir predicados tanto de igualdad como de rango. Finalmente, con el fin de considerar la más amplia gama de consultas posibles sobre los atributos de la relación EMP, cada tipo de consulta Q_i incluye todas las posibles combinaciones de atributos. Así por ejemplo, el tipo Q_1 se compone de 100 consultas de selección sobre el atributo *Ciudad*, 100 consultas sobre el atributo *Edad*, y así sucesivamente.

Con objeto de verificar el funcionamiento correcto del árbol Q como método de búsqueda, las consultas incluidas en todas las baterías diseñadas se ejecutaron previamente utilizando la base de datos *Access para Windows*. Para asegurar la validez de nuestras evaluaciones posteriores, los resultados derivados a partir de esta aplicación fueron posteriormente confrontados con aquellos que proporcionó el árbol Q.

2.3 Medidas de rendimiento

2.3.1 Sensibilidad al número de dimensiones

Como se describió en el apartado anterior, el objetivo de evaluación para el cual se diseñó la Batería 1 consistía en comprobar el comportamiento de la estructura de árbol Q frente a la variación en el número de atributos indexados o, lo que es lo mismo, frente a un número variable de dimensiones.

Para medir este comportamiento, nuestros patrones de referencia se centran en el grado de utilización de las páginas físicas tanto en el índice como en los datos y, adicionalmente, en conocer la tendencia en el crecimiento del índice a medida que aumenta el número de dimensiones.

La cantidad de espacio útil de las páginas físicas es una medida de referencia habitual de las estructuras dinámicas equilibradas similares al árbol B, en las que la reorganización es completamente dinámica y se lleva a cabo a medida que van llegando nuevas tuplas a la relación. En este sentido, la figura 5-1 muestra el grado de utilización del espacio en árboles Q para un variado número de dimensiones.

Dos conclusiones inmediatas pueden obtenerse de la observación de las gráficas exhibidas en la figura 5-1:

- El grado de utilización del espacio físico, tanto a nivel de contenedores (datos) como a nivel de regiones y espacios (índice), no depende del número de dimensiones indexadas por el árbol Q.
- Teniendo en cuenta el grado medio de ocupación de nodos en árboles B+, el cual es de ln2 (el 69% de ocupación aproximadamente), el nivel de ocupación del árbol Q se mueve en una horquilla (66 - 70% aproximadamente) ciertamente comparable a la de aquél.

Respecto a la primera de las conclusiones, conviene reseñar que las diferencias en el grado de ocupación a ambos niveles (datos e índice) se deben básicamente al tipo de los atributos que se incorporan gradualmente al árbol, en los que, tanto el tamaño de su dominio (número de valores distintos del atributo), como el atributo utilizado para la división de un contenedor, determinan de forma distinta el contenido de cada uno de los contenedores producto de la división. Teniendo en cuenta que el contenido de las páginas de datos va dictando la forma del índice, es claro que la división de los contenedores tiene una influencia directa en la división de regiones y espacios y, por consiguiente, en la ocupación de las páginas.

Los menores porcentajes de utilización en el índice respecto a las páginas de datos se derivan, fundamentalmente, del grado mínimo de ocupación exigido en los Q-nodos de la zona de índices que, como sabemos, es de 1/3 de la capacidad de la página. Recordemos que el nivel de ocupación mínimo de los contenedores es ajustable por el usuario y, en nuestros experimentos, lo hemos fijado en un 40% de la capacidad del contenedor. Uniendo a esto el hecho de que el Q-nodo raíz puede llegar a contener un árbol k-d local con sólo tres nodos, y que la frecuencia con la que se van llenando las páginas del índice es menor a medida que subimos en altura, se explica con mayor claridad las diferencias existentes entre las gráficas (a) y (b) de la figura 5-1.

Respecto a la posible tendencia del crecimiento del índice en relación al número de atributos indexados, la figura 5-2 muestra los resultados de nuestros análisis. Como bien puede apreciarse, existe una tendencia creciente, que se hace más suave mientras mayor es el tamaño considerado de la página física. Dado que el contenido básico de los Q-nodos índice está constituido por un árbol k-d local, en el que cada nodo almacena el valor de un atributo, resulta evidente que, a diferencia de otros métodos de indexación multiatributo que albergan información de fronteras multidimensionales, tales como el árbol R [32], el árbol Q no tiene

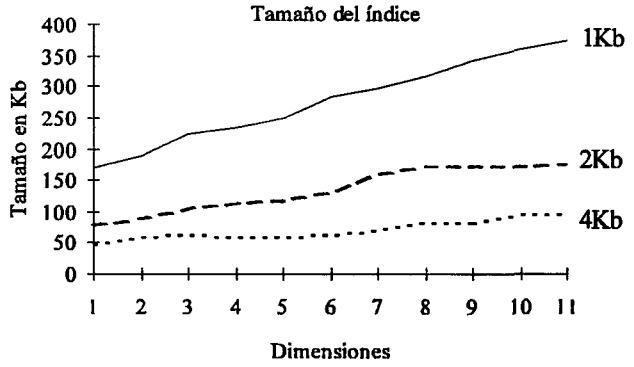


Figura 5-2. Tendencia de crecimiento del índice según los atributos indexados.

por qué verse fuertemente afectado por la variación en el número de dimensiones. Sin embargo, en la versión implementada del árbol Q, los nodos índice almacenan cierta información de control en forma de vectores con tantos elementos como atributos indexados. Estos elementos provocan dicha tendencia alcista en el tamaño del índice. Lógicamente, mientras mayor es el tamaño de la página, menor es el ratio entre el espacio utilizado para esta información de control y el resto de la información propia del índice. Este hecho se refleja en una influencia prácticamente nula del número de dimensiones con tamanos de página de 4 Kb o mayores.

A diferencia de otros métodos de indexación multiatributo, conviene resaltar el mejor rendimiento que reporta el árbol Q. En los ficheros rejilla [52], por ejemplo, cada dimensión requiere un vector (llamado escala) con tantos componentes como rangos exiten para dicha dimensión. Este espacio extra puede no ser extremadamente voluminoso si los datos se distribuyen uniformentente. Sin embargo, ante la presencia de fuerte sesgo o correlación entre los atributos indexados, los requerimientos de espacio pueden crecer exponencialmente con el número de dimensiones.

Los árboles hB^{Π} [22], por su parte, almacenan la información de las fronteras de cada región del espacio del cual un nodo índice es responsable. Lógicamente, el espacio que ocupa esta información (que precisa de dos valores por cada dimensión indexada) crece en un factor de dos con el número de dimensiones. En el caso de atributos cuyo tipo requiera un número elevado de bytes, la tendencia creciente del tamaño del índice puede llegar a ser pronunciada a medida que aumenta el número de dimensiones.

Merece destacar por último que, para tamaños de página de 4Kb, el tamaño que alcanza el índice supone un bajo porcentaje del tamaño real de los datos (2.7% por término medio aproximadamente), lo que confiere al método de acceso en árbol Q una característica, deseable en las estructuras índice, de manejabilidad y adaptabilidad a la fuerte competencia por los recursos de memoria principal típica de las aplicaciones actuales. Teniendo en cuenta que a

partir de nuestras experiencias, con el mismo conjunto de datos, en una estructura de árbol B^+ la zona índices representa aproximadamente el 1,25% del tamaño real de los datos, nos movemos en unos niveles razonables de ocupación para estructuras multiatributo.

2.3.2 Rendimiento frente a consultas

Con objeto de evaluar el comportamiento del árbol Q como método de acceso, el conjunto de pruebas incluídas en la Batería 2 ha sido ejecutado en dos ocasiones sobre sendas implementaciones del árbol Q. En una de ellas se utilizó la estrategia de selección por niveles y, en la segunda implementación, utilizamos la estrategia de selección del mejor atributo. Los criterios de evaluación tomados como referencia en el desarrollo de nuestros experimientos son dos:

- porcentaje de acceso a datos, y
- ratio de aciertos sobre los contenedores consultados, es decir el cociente, medido en términos porcentuales, entre el número de tuplas que satisfacen el predicado y el total de tuplas examinadas.

Los resultados para estos parámetros se obtienen lógicamente en función de la selectividad de las consultas de cada uno de los cinco tipos que se incluyen en la Batería 2.

La figura 5-3 muestra los resultados relativos al porcentaje de acceso a datos respecto al factor de selectividad de las consultas. Los valores mostrados en las gráficas se obtienen como media de los resultados obtenidos a partir de todas aquellas consultas en el tipo correspondiente, que tienen como factor de selectividad el indicado. Una gráfica para cada uno de los tipos de consulta definidos se presenta en dicha figura.

Varias conclusiones pueden extraerse a partir de la observación de las gráficas en la figura 5-3:

- A medida que se complica la consulta, incrementando el número de atributos que aparecen en el predicado de la misma, el método de acceso realiza una mayor discriminación en el conjunto de datos, tal y como debe exigirse a un método eficiente de búsqueda multiclave.
- El árbol Q mantiene un nivel de discriminación razonable aún en el caso de ser utilizado para responder consultas en las que sólo se interroga acerca de una pequeña porción de los atributos indexados. La gráfica para Q_1 muestra una relación ligeramente superlineal a partir de un factor 20 de selectividad entre el número de tuplas en la respuesta y el porcentaje de datos accedidos.
- Teniendo en cuenta el sesgo que presentan los datos almacenados para algunos de los atributos indexados, un porcentaje máximo de acceso al 4.5% (en Q_5) del espacio de búsqueda global cuando se plantean consultas de selección arbitrariamente

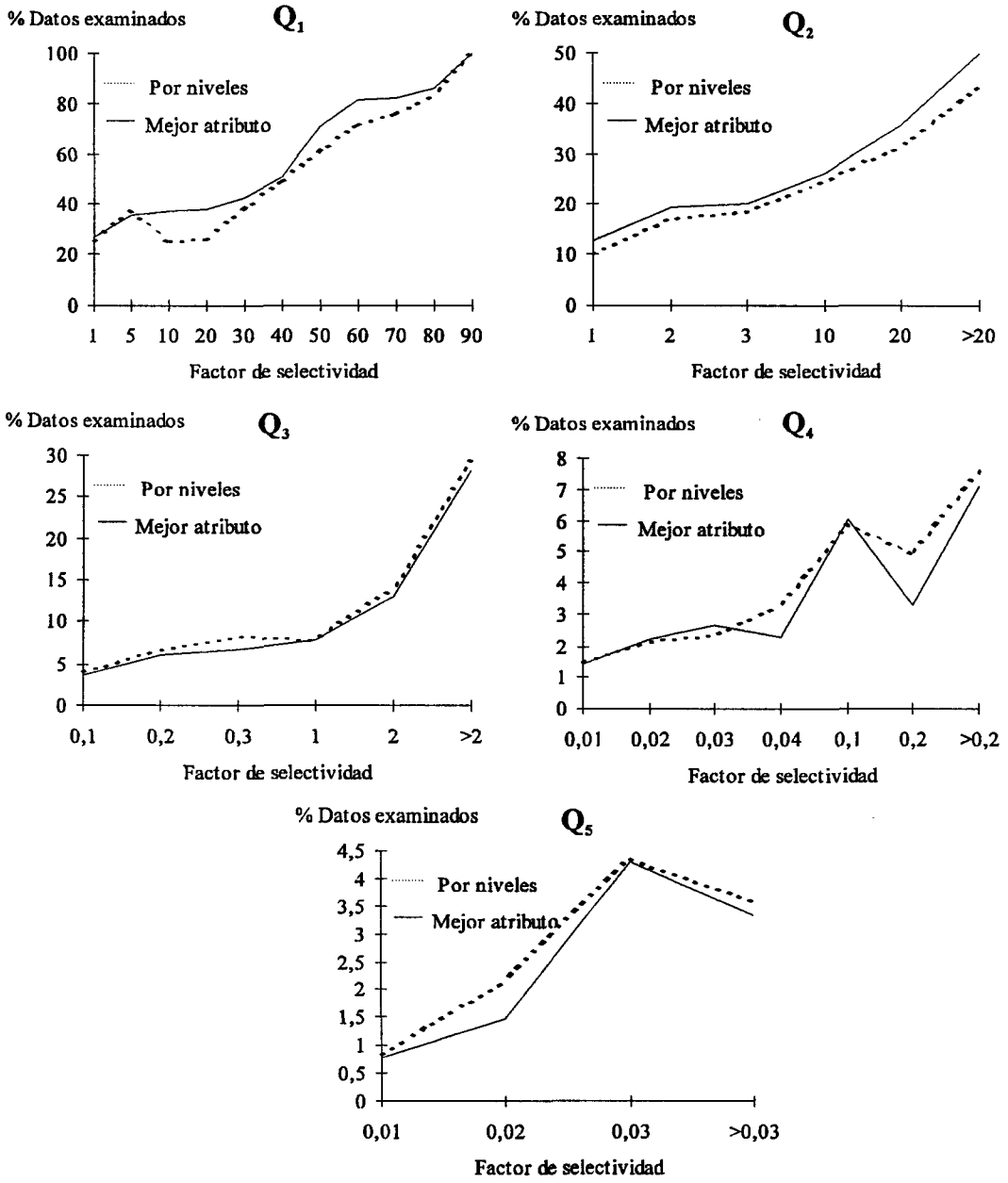


Figura 5-3. Porcentaje de acceso a datos para cada tipo de consulta en la Batería 2.

complejas (restringidas por supuesto a predicados de tipo conjuntivo), equivale a una reducción *considerable* del dominio de búsqueda examinado, en comparación a los métodos de acceso en los sistemas actuales de bases de datos[2].

2 Desafortunadamente, la falta de estudios similares en otras estructuras multiatributo no nos permiten realizar una comparación realista, que nos lleve a comprender en toda su amplitud los resultados obtenidos.

- A medida que se incorporan más atributos a las consultas, el método de selección del mejor atributo mejora progresivamente el comportamiento de la estrategia de selección por niveles. Parece claro que, en la previsión de utilizar en las consultas un bajo porcentaje de los atributos indexados, es preferible la estrategia de selección por niveles. Por contra, si el porcentaje de atributos referidos en el predicado de las consultas tiende a acercarse al de atributos indexados, el árbol Q construido con la estrategia de selección del mejor atributo, obtiene mejores resultados.

Para mostrar la eficiencia del método en cuanto a los aciertos obtenidos, se pueden considerar dos fórmulas diferentes. Una de ellas consistiría en comprobar el número de tuplas que satisfacen el predicado en relación al número total de tuplas procesadas. Este método sería fiable si el coste de procesar dos tuplas cualesquiera fuera el mismo. Tal es el caso en que los datos se mantuvieran en memoria principal. Sin embargo, cuando para procesar una tupla es preciso realizar accesos a disco, tal hipótesis deja de ser válida, ya que el coste de procesar dos tuplas varía en función de si residen en el mismo o diferente contenedor.

Otro modo distinto de medir el porcentaje de aciertos consiste en calcular el acceso a contenedores, en lugar de examinar el coste del proceso de tuplas. En este caso se obtendría el cociente entre el número de contenedores "válidos" y el número total de contenedores accedidos. Un contenedor se considera "válido" cuando almacena al menos una tupla que satisface el predicado. Aunque esta fórmula no adolece de los problemas de la anterior presenta, por contra, fuertes dependencias del tamaño que se elija para el contenedor. Para explicar esto, basta considerar el caso extremo de que un contenedor tenga tamaño suficientemente para que todo el espacio de búsqueda pueda almacenarse allí. Bajo este escenario, siempre obtendríamos un porcentaje del ciento por cien, lo cual no dice absolutamente nada a favor ni en contra del método de acceso cuando el factor de selectividad es bajo.

Las anteriores consideraciones nos conducen a una clara reflexión al respecto: teniendo en cuenta que el volumen de datos es suficientemente grande como para precisar de accesos a disco en la búsqueda de tuplas que satisfacen un predicado, el porcentaje de aciertos medido en términos de contenedores se considera la elección más idonea. Dado que la figura 5-3 muestra los porcentajes de acceso a datos, el declarar cúantas de las páginas de datos accedidas contienen tuplas relevantes, nos aporta una información interesante desde el punto de vista de la utilidad de los accesos realizados.

La figura 5-4 muestra los resultados obtenidos utilizando el porcentaje de aciertos en el acceso a contenedores, para cada tipo de consulta en la Batería 2. Como es obvio, un porcentaje del 100% revela que cada contenedor accedido contenía alguna tupla satisfaciendo el predicado y, por consiguiente, que ningún acceso resultó inútil, mostrando el árbol Q en este caso un rendimiento óptimo. Por contra, mientras menor sea este porcentaje, peor rendimiento aporta la estructura en la respuesta a una consulta.

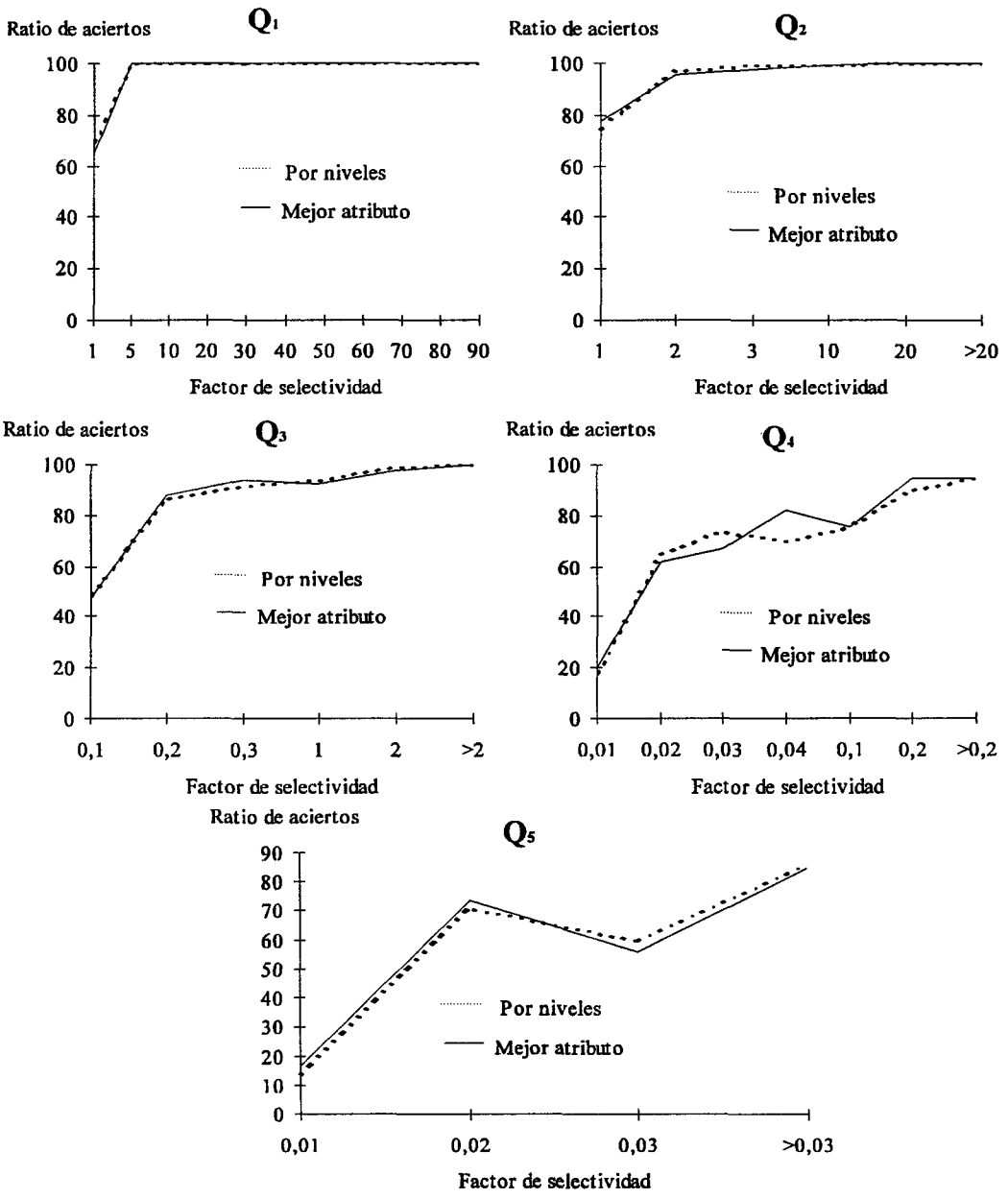


Figura 5-4. Ratios de acierto en el acceso a contenedores.

Para consultas de tipo Q_1 que, como vimos por la figura 5-3, consigue unos porcentajes de acceso a datos ciertamente buenos, la gráfica en la figura 5-4 nos dice además que prácticamente todos los accesos, a partir de un factor de selectividad del 5%, tuvieron éxito. Podemos concluir de esta observación que, en el caso de consultas por un sólo atributo, el árbol Q discrimina la búsqueda de forma casi perfecta, en función de cómo está distribuido el espacio.

A partir de la figura 5-4 se observa también que, a medida que aparecen más atributos en el predicado de la consulta (cuando pasamos de una consulta de tipo Q_i a otra de tipo Q_{i+1}), los porcentajes de acierto son cada vez menores y, adicionalmente, la disminución es más patente cuando menor es el factor de selectividad de la consulta.

Los predicados que incluyen un alto número de atributos resultan en consultas complejas con muy pocas tuplas en el resultado (muestra de ello son los factores pequeños de selectividad que muestran la gráficas correspondiente a Q_4 y Q_5). Esto no significa necesariamente que la región multidimensional que define el predicado sea pequeña, sino que es posible que el sesgo que presentan los datos favorezca la inexistencia de tuplas relevantes dentro de los contenedores que intersecan la región buscada.

Lo importante es que cuando el factor de selectividad es bajo se accedan a pocos contenedores. Si además de ello, una alta proporción de tales contenedores almacena tuplas que satisfacen el predicado, esto significa que la distribución del espacio que consigue el método de búsqueda es correcta (no deseamos acceder a contenedores para descubrir que no contienen tuplas relevantes).

Estudiando al unísono las gráficas correspondientes a cada tipo de consulta dentro de las figuras 5-3 y 5-4 podemos concluir que nuestra estructura índice en árbol Q, ofrece una respuesta eficaz ante la búsqueda de datos, independiente del número de atributos que aparecen en el predicado. A modo de ejemplo, basta considerar las gráficas correspondientes a Q_4. La figura 5-4 demuestra que el porcentaje de aciertos para un factor de selectividad del 0.02 es próximo al 65%. Esto significa que podríamos habernos ahorrado un 35% de accesos a contenedores. La cuestión ahora es ¿qué coste representa este gasto inútil?. Teniendo en cuenta la figura 5-3, la gráfica correspondiente a Q_4 indica, para el mismo factor de selectividad, que se accedió a algo más de un 2% de los contenedores. Conjugando ambos resultados, el trabajo inútil supone sólo el acceso a un 0.7% del total de contenedores en el fichero de datos. Tal y como se comentó con anterioridad, este porcentaje puede considerarse razonable teniendo en cuenta que las estructuras de acceso multiclave se enmarcan dentro de métodos de búsqueda cuya finalidad es la de **restringir** en lo posible el número regiones del espacio (contenedores en nuestro caso) que es preciso examinar para obtener la respuesta solicitada.

Para finalizar este apartado de evaluación del árbol Q, exponemos a continuación los aspectos más sobresalientes de la estructura propuesta, a tenor de las pruebas realizadas:

- Presenta grados de utilización del espacio acorde con las estructuras índices de reorganización dinámicas, tales como árboles B+.
- Resulta muy poco sensible al número de dimensiones indexadas, lo que le confiere de propiedades deseables para el tratamiento de datos espaciales y, en general, de datos cuya información se almacene en término de un gran número de atributos interesantes desde el punto de vista de la búsqueda.

- El índice ocupa poco espacio en relación a los datos, lo que favorece su almacenamiento, al menos en parte, dentro de la memoria principal, evitando así un mayor número de accesos.
- Presenta grados aceptables de discriminación en las búsquedas, lo que redunda en porcentajes de acceso consecuentes con el número de tuplas que satisfacen la respuesta.
- Revela un buen comportamiento en el tratamiento de datos no uniformes, con presencia de sesgo en la distribución de sus valores.
- Demuestra niveles de rendimiento razonables cualquiera que sea el número de atributos incluidos en el predicado de la consulta. Esto permite al árbol manejar de forma equivalente consultas con grados de complejidad variable.

En el siguiente apartado nos centramos en examinar los resultados obtenidos cuando la estructura de árbol Q se utiliza como soporte del almacenamiento de relaciones en un SBDP.

3 Evaluación del particionamiento multidimensional basado en árbol Q

En este apartado presentamos el resultado de las pruebas de medición efectuadas sobre el árbol Q, respecto a su explotación como soporte al particionamiento multidimensional en máquinas paralelas de bases de datos. Siguiendo el mismo esquema que en el apartado previo, describimos inicialmente la batería de pruebas construidas al efecto de la evaluación de los diferentes mecanismos definidos en el capítulo 4 y, posteriormente, entramos a valorar los resultados obtenidos.

3.1 Batería de pruebas

Los principales objetivos en la fase de evaluación del rendimiento de nuestra propuesta de particionamiento multidimensional pueden sintetizarse en los siguientes puntos:

1 Comparar el rendimiento del particionamiento multidimensional basado en el uso del árbol Q, con otras estrategias de particionamiento basadas en el uso de un sólo atributo (las cuales denominamos estrategias de particionamiento lineales).

2 Examinar las diferentes alternativas de particionamiento basado en árbol Q, con el fin de establecer las potencialidades y debilidades que exhiben dentro del contexto en el que son aplicadas.

3 Verificar la utilidad del particionamiento multidimensional basado en árbol Q en máquinas paralelas de memoria distribuida respecto a criterios de aceleración y escalabilidad.

Con tales objetivos en mente, hemos construido un nuevo conjunto de pruebas "a medida", compuesto por múltiples consultas de selección con predicados de coincidencia exacta, coincidencia parcial y predicados de rango sobre dos atributos diferentes con valores no únicos.

En este contexto, conviene destacar la dificultad que encontramos a la hora de utilizar alguna de las baterías de dominio público que evalúan el rendimiento de sistemas de bases de datos. Aunque se han diseñado múltiples conjuntos de datos y consultas para la exploración de sistemas paralelos, tales baterías persiguen el objetivo genérico de medir el comportamiento del sistema en su globalidad. Con objetivos tan específicos como los que nos atañen, tales elementos de medida tienden a ocultar importantes detalles debido a su generalidad.

En el marco específico de los sistemas paralelos de bases de datos, la batería de pruebas más ampliamente aceptada es la conocida como el *Wisconsin Benchmark* [29]. En un primer acercamiento a la evaluación de nuestros esquemas de particionamiento, estuvimos tentados de utilizar dicha batería de pruebas debido a la facilidad con la que podríamos extrapolar los resultados obtenidos. Sin embargo no tardamos en comprender que, con un modelo de ejecución tan sencillo como el que se describió en el capítulo 4 y sin la existencia de una máquina de base de datos simulada en una gran parte, los resultados no iban a ser tan significativos como esperábamos. Por otra parte, teniendo en cuenta las características no convencionales de nuestro modelo de particionamiento, en el que resultaba de gran interés medir su comportamiento ante la presencia de sesgo en los datos y fuerte correlación entre distintos atributos, los datos fuente proporcionados por el test de *Wisconsin*, así como las consultas de selección del mismo, no ofrecían posibilidades para extraer conclusiones a este respecto.

La falta pues de especificidad de las más típicas pruebas para sistemas de bases de datos paralelas existentes en la literatura, nos llevó a la decisión de plantear nuestro propio conjunto de pruebas de evaluación.

Decididos a construir una batería específica propia, la base de datos inicial consiste en una relación *EMP* con 50.000 tuplas, cuyo esquema de relación incluye seis atributos, dos de los cuales, denominados *Edad* y *Salario*, son claves secundarias sobre cuyos valores se emite un conjunto de consultas.

Para capturar el efecto del sesgo en los datos, los valores de *Edad* se generan de acuerdo a un vector de probabilidad que favorece la aparición de ciertos valores en discriminación de otros. Además, puesto que la correlación entre atributos de una relación es un factor que aparece con frecuencia en aplicaciones típicas de bases de datos, los valores del atributo *Salario* se han generado de modo que exista una alta correlación con el valor sobre el atributo *Edad* (normalmente el salario de un empleado aumenta a medida que dicho empleado es mayor, al menos parece lógico que así sea).

Tabla 5-3. Descripción del conjunto de consultas para la Batería 3.

Consulta	Predicado	Atributo	Frecuencia de emisión	Factor de selectividad
Q_1	Igualdad	Edad	10%	8.7%
Q_2	Rango	Edad	15%	28%
Q_3	Rango	Edad	15%	13%
Q_4	Rango (AND)	Edad y Salario	60%	0.5%

Además, la relación *EMP* contiene una clave primaria única que identifica a las tuplas de la relación y cuyos valores se utilizan en la fase de creación del árbol Q.

Para validar la utilidad de las diferentes estrategias bajo escenarios diferentes, hemos construido cuatro tipos de consultas (Q_1 a Q_4) de acuerdo a la distribución de frecuencias y a los factores de selectividad mostrados en la tabla 5-3.

La batería de pruebas global se compone finalmente de un total de 1000 consultas de los tipos Q_1 a Q_4 con las frecuencias referidas. Los valores que aparecen en el predicado de las consultas siguen aproximadamente la misma distribución que los valores de las tuplas en la relación. En el caso de predicados rango, dichos rangos varían en tamaño de consulta a consulta y fueron generados como sigue. Para cada dimensión específica, se determinó el centro del rango en función de la distribución del dominio para dicha dimensión. Posteriormente, se obtuvieron aleatoriamente las longitudes de los rangos, utilizando para ello una función aleatoria de distribución uniforme. A esta batería específicamente diseñada la denominamos Batería 3.

Los resultados derivados a partir de esta batería de pruebas son comentados en el siguiente apartado.

3.2 Medidas de rendimiento

3.2.1 Particionamiento multidimensional frente al particionamiento lineal

Para comparar el rendimiento del particionamiento multidimensional basado en árbol Q con modelos más clásicos de particionamiento en bases de datos paralelas, hemos implementado una estrategia de fragmentación de relaciones que ha aportado buenos resultados en el marco de la máquina de base de datos Bubba [13]. La estrategia de particionamiento a que nos referimos recibe el nombre de *BERD* ("*Bubba's extended-range declustering*"), esto es, estrategia de fragmentación de rango extendido en Bubba.

Utilizaremos un ejemplo sencillo con el fin de ilustrar la forma en la que BERD particiona una relación. Consideremos una relación R con dos atributos A y B, y una cardinalidad de seis tuplas. BERD distingue entre un atributo de particionamiento primario y uno secundario,

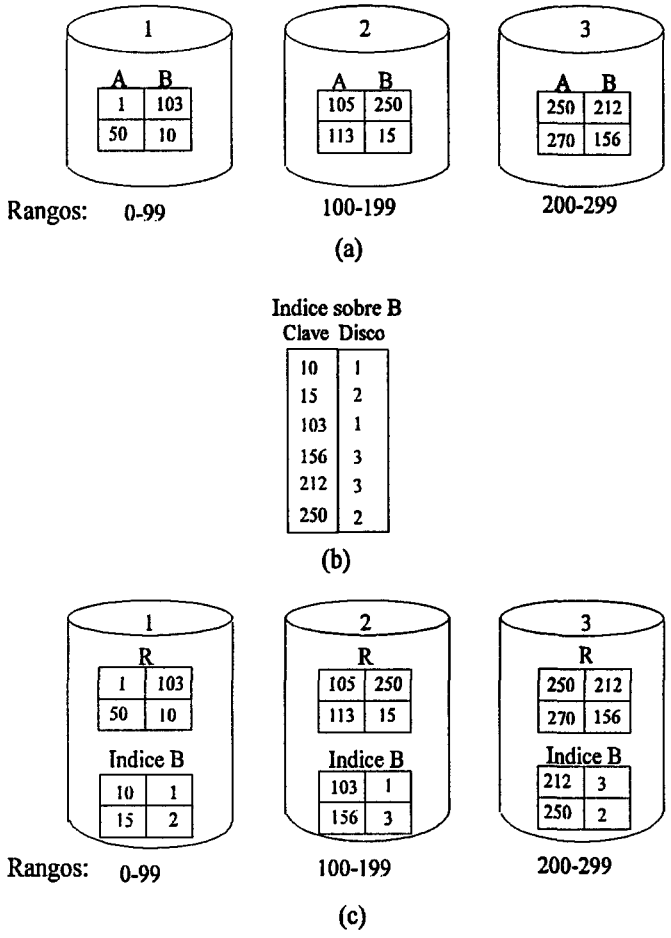


Figura 5-5. Ejemplo de particionamiento según la estrategia BERD.

fragmentando la relación R por rangos usando los valores del atributo primario. Suponiendo que el atributo primario es *R.A*, la relación *R* se fragmenta por rangos sobre el valor de *A*, según muestra la figura 5-5a.

A continuación se construye un índice por el atributo secundario B, que nos informe sobre la asignación de los distintos valores de B que llevó a cabo el particionamiento primario realizado previamente. Este índice se muestra en la figura 5-5b.

Finalmente, el índice sobre B se particiona por rangos equivalentes según el valor de B y se asigna a los procesadores implicados, tal y como muestra la figura 5-5c. En este caso "rangos equivalentes" significa exactamente el mismo rango para ambos atributos. Un directorio a disposición del procesador de consultas indicará los rangos para los atributos primario y secundario y su asignación a correspondientes procesadores.

Con esta estrategia de particionamiento, si se emite una consulta con un predicado en rango especificando el valor del atributo primario A, el módulo procesador de consultas utiliza la información de particionamiento de este atributo para dirigir la consulta a aquellos

procesadores con fragmentos relevantes de la relación. Si la consulta se especifica en términos del atributo secundario B, se examina la información de directorio correspondiente a los rangos de B. Esta información nos conduce a los procesadores relevantes donde se almacena el índice B, a partir del cual podemos obtener los procesadores donde debe dirigirse la consulta. Como se deriva del ejemplo anterior, el número real de procesadores usados en la ejecución de una consulta depende de la correlación existente entre los atributos A y B. Mientras más baja sea la correlación entre ambos atributos más procesadores participarán en la ejecución de las consultas.

Para simplificar el trabajo, en la implementación que hemos efectuado de esta estrategia, hemos considerado que el índice sobre el atributo secundario es una información global que no se encuentra fragmentada. Esto ahorra una cierta cantidad de indirección que sería necesaria efectuar en el caso de particionar el índice.

De acuerdo a los datos de nuestra batería de pruebas, la estrategia BERD se implementó por partida doble. En una ocasión se consideró el atributo *Edad* como atributo primario y *Salario* como secundario, y en la otra se invirtieron los papeles de ambos atributos.

Para implementar BERD, se construyó un árbol B+ para el atributo primario, que particionase por rangos los valores de dicho atributo. Posteriormente, se ajustaron los rangos para fragmentar equilibradamente el conjunto de bloques de datos (conjunto secuencia del árbol B+) entre el conjunto de procesadores. Por su parte, el índice secundario se construyó por medio de un índice denso en árbol B, que nos aportase la información necesaria para conocer la asignación de las tuplas según los valores del atributo secundario.

Nótese que a pesar de utilizar más de un atributo para el particionamiento de las relaciones, en realidad, la estrategia BERD es un método de particionamiento lineal (a diferencia del particionamiento multidimensional). Cuando se emite una consulta en cuyo predicado

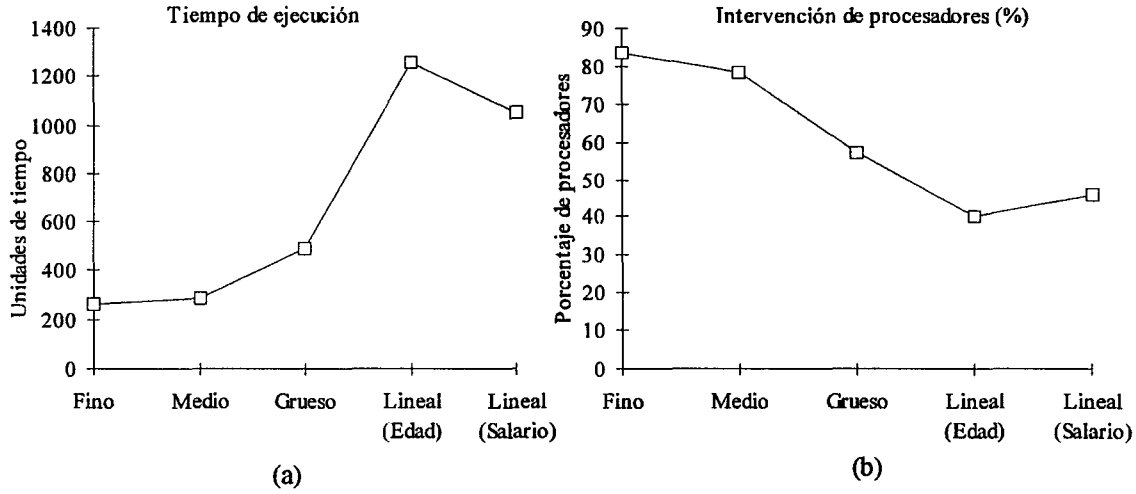


Figura 5-6. Rendimiento del particionamiento multidimensional frente al particionamiento lineal. (a) Tiempo medio de ejecución. (b) Porcentaje de intervención de procesadores.

aparecen ambos atributos de particionamiento, BERD utiliza en exclusividad la información de particionamiento primario para dirigir la consulta hacia los procesadores relevantes.

En los experimentos que se describen a lo largo de este apartado, y mientras no se indique lo contrario, hemos considerado que la máquina simulada, soporte de dichos experimentos, consiste en un total de ocho procesadores.

La figura 5-6 demuestra un rendimiento mucho mejor del particionamiento multidimensional frente al particionamiento lineal en las dos vertientes analizadas. En la figura 5-6a se exhiben los tiempos medios de respuesta, medidos a partir del total de consultas de la Batería 3. Los tiempos de respuesta se exhiben para cada una de las tres propuestas BEPI en comparación a las dos implementaciones establecidas sobre el modelo de particionamiento lineal descrito previamente. Debido a la simplificación del modelo de ejecución de consultas descrito en el capítulo anterior, los valores de tiempo que mostramos en las gráficas se consideran como unidades genéricas de tiempo, cuyo valor no es significativo fuera del ámbito en el que se obtiene.

En el caso medio, la estrategia de particionamiento multidimensional con peor comportamiento (estrategia BEPI de grano grueso), consigue una reducción del 50% en los tiempos de respuesta respecto a la estrategia de particionamiento lineal que observa mejor comportamiento (atributo primario *Salario*).

Las diferencias de rendimiento entre el particionamiento lineal y multidimensional se comprenden mejor examinando, adicionalmente, el porcentaje de procesadores involucrados en la ejecución de las consultas de la Batería 3. La figura 5-6b refleja que el porcentaje promedio en el uso de procesadores cuando se utiliza el particionamiento multidimensional es significativamente mayor que en el caso de utilizar particionamiento lineal.

La utilización de un alto porcentaje de los recursos de una máquina paralela no es, sin embargo, orientativo de la eficacia del método. Es preciso además valorar la forma en que se

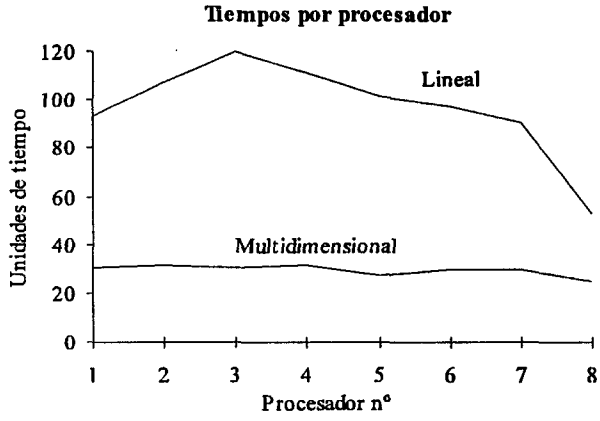


Figura 5-7. Tiempo invertido por cada procesador en la estrategia multidimensional frente a la estrategia de particionamiento lineal.

utilizan tales recursos. En este sentido, la figura 5-7 demuestra que el método de particionamiento multidimensional, en comparación con el modelo lineal, consigue mejor equilibrio en el consumo de tales recursos. Esta figura muestra los tiempos medios invertidos por cada uno de los ocho procesadores de la máquina simulada, tanto en el caso multidimensional (cuyos valores se obtienen como media de las tres alternativas BEPI), como en el caso lineal (obtenido como media de las dos configuraciones implementadas para este caso). Los tiempos mostrados son, así mismo, promedio de todas las consultas en la Batería 3.

Como se desprende de la figura 5-7, el particionamiento lineal no sólo conlleva unas mayores inversiones de tiempo por cada procesador en la máquina, sino que además el tiempo invertido varía en una franja considerablemente ancha de procesador a procesador. Por el contrario, la estrategia de particionamiento multidimensional consigue tiempos de dedicación en los procesadores que se mantienen casi constante en todos los procesadores. Esta última propiedad del modelo de particionamiento multidimensional basado en árbol Q, revela una característica importante en el ámbito del paralelismo: equilibrio de carga entre procesadores, aún en presencia de sesgo en los datos. Aunque este última gráfica no muestra valores absolutos para una estrategia específica, resulta orientativa para entender que el particionamiento multidimensional resulta ser, como poco, una prometedora estrategia para fragmentar relaciones en bases de datos paralelas.

En los siguientes subarpartados, una vez alcanzado el primero de los objetivos de evaluación propuestos, examinamos en detalle el rendimiento de las diferentes estrategias de particionamiento multidimensional basadas en el uso del árbol Q.

3.2.2 Evaluación de estrategias respecto al equilibrio de carga

Una vez adelantados los potenciales beneficios que pueden obtenerse con la aplicación de enfoques multidimensionales para el particionamiento de relaciones, es necesario evaluar en detalle el grado de efectividad para cada uno de los métodos de particionamiento propuestos en la presente tesis.

Con tal objeto comenzamos la descripción de nuestros resultados comparando las tres estrategias BEPI propuestas para, posteriormente, confrontar el modelo BEPI frente al modelo BCCT.

Comparación de las estrategias BEPI

En primer lugar haremos la comparación de los distintos mecanismos sobre la base de un número fijo de ocho procesadores en la máquina multicomputador simulada. Para ello comenzamos este análisis mostrando resultados relativos a tiempos de respuesta y grado de equilibrio en la distribución de datos de la relación *EMP* entre los ocho nodos, utilizando las

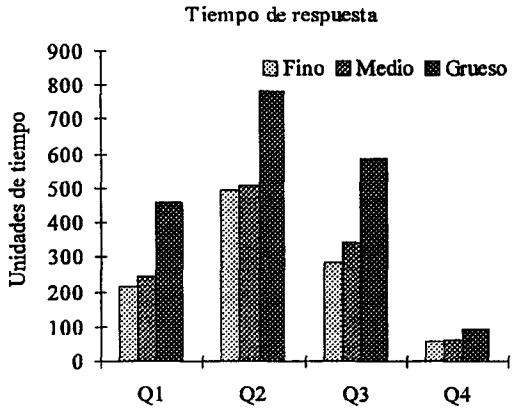


Figura 5-8. Comparación de los tiempos de respuesta en las estrategias BEPI.

estrategias BEPI. En este sentido, la figura 5-8 sirve como punto de partida de nuestro análisis. En esta figura se muestran los tiempos de respuesta para cada tipo de consulta de la Batería 3 invertidos en cada uno de los enfoques de particionamiento BEPI.

Algunas particularidades pueden extraerse de esta figura. Primeramente y como conclusión general podemos deducir que la estrategia de grano fino presenta los mejores resultados para todos los tipos de consulta. Además, la figura también refleja la forma efectiva en la que el árbol Q ayuda a responder las consultas de la Batería 3 según su tipo. Como puede verse, todas las alternativas de particionamiento multidimensional BEPI son capaces de capturar la diferencia en los factores de selectividad de los distintos tipos de consulta (Tabla 5-3), mostrando tiempos de ejecución en consonancia con tales diferencias.

En una primera aproximación y a la vista de la figura 5-8, podríamos concluir que el grano fino es la estrategia candidata a ser elegida como método de particionamiento basado en árbol Q. Sin embargo, resulta interesante conocer cómo se comportan las distintas estrategias frente a criterios de equilibrio de carga de los procesadores. Considerando que, respecto al rendimiento global del sistema ("*throughput*"), es fundamental conseguir una distribución equilibrada de la carga de trabajo (aún a expensas de unos mayores tiempos de respuesta de una típica consulta), resulta indispensable pues, examinar el comportamiento de las estrategias BEPI en función dichos criterios de equilibrio.

Las gráficas en la figura 5-9 exhiben los tiempos individualmente invertidos por cada procesador en el sistema durante la ejecución todas las consultas en la Batería 3, distribuidos por tipo de consulta. En ellas se pueden apreciar diferencias de tiempo invertido por algunos procesadores específicos (normalmente distintos para cada estrategia considerada), que demuestran la existencia de zonas *calientes* del espacio de búsqueda, es decir, zonas del espacio que son accedidas con más frecuencia que otras.

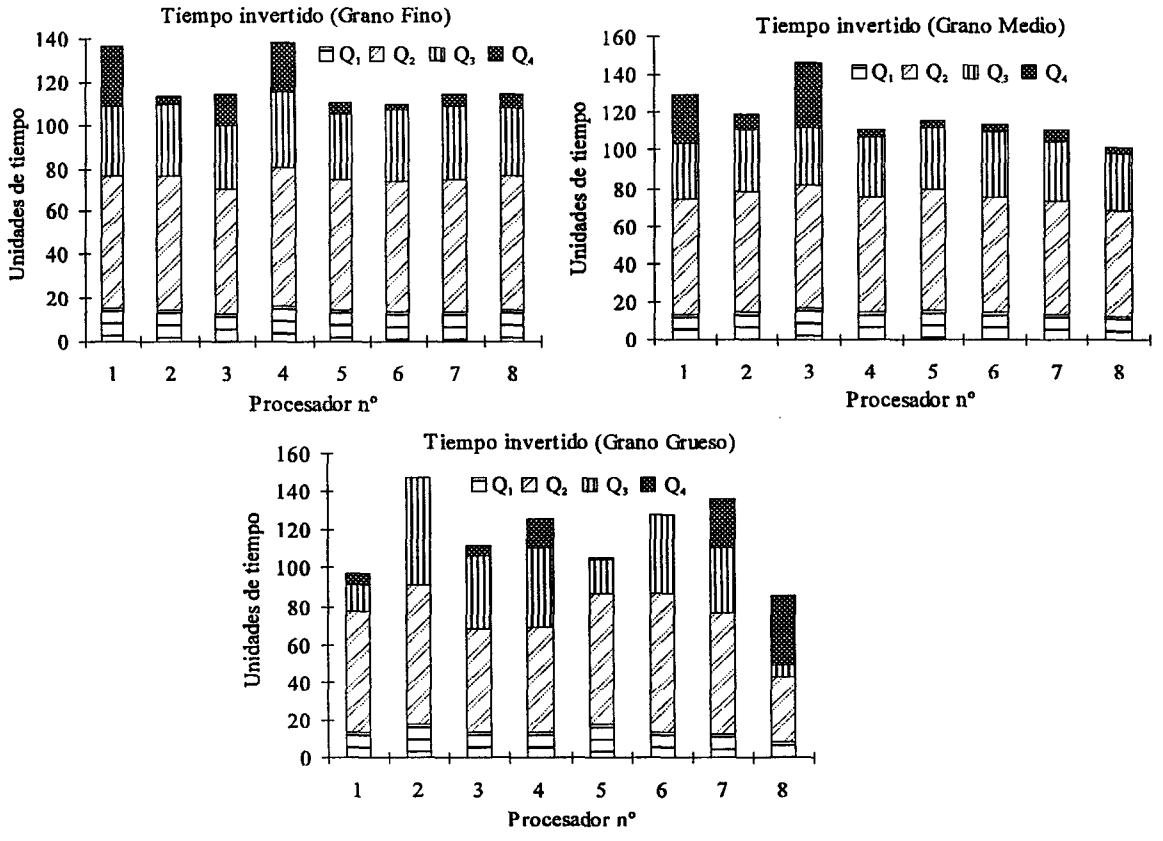


Figura 5-9. Tiempo invertido por procesadores individuales en las estrategias BEPI.

Sin embargo, para extraer conclusiones al respecto del rendimiento global del sistema, interesa centrarse en la manera de repartir la carga de trabajo entre todos los procesadores. Un buen comportamiento supondría que todos los procesadores dedicaran aproximadamente la misma cantidad de tiempo, no sólo desde la perspectiva de la carga global del sistema, es decir, considerando el trabajo global a que es sometido el sistema, sino también desde un punto de vista más local, esto es, sometiendo a los procesadores a unidades más pequeñas de trabajo. Esto es así debido a que la carga de trabajo de un sistema no es, por lo general, uniforme. En particular, las pruebas de nuestra batería no requieren todas la misma inversión de recursos, sino que se compone de una mezcla sopesada de distintos tipos de consulta con requerimientos diferentes. Un sistema equilibrado debería comportarse con el mismo grado de equilibrio tanto desde el punto de vista global (una vez ejecutada la batería completa de consultas), como desde una perspectiva más parcial (observando su comportamiento ante los diferentes tipos de consultas).

En este sentido, lo más reseñable de la figura 5-9 es que, para todas las estrategias de particionamiento BEPI consideradas, todos los procesadores cooperan en la ejecución de las consultas. Sin embargo, en lo que respecta al equilibrio de carga del sistema, la situación no es tan regular. Una estrategia de particionamiento ideal distribuiría datos de tal modo que los ocho procesadores dedicarían el mismo tiempo a las consultas de cada uno de los tipos

considerados. Pero como bien hemos dicho, esto es una situación **ideal**. En un escenario real, la carga de trabajo no es una característica estática del sistema y puede variar en el tiempo. Algunas partes del espacio de búsqueda pueden ser requeridas con más frecuencia que otras en determinados momentos, de modo que, a menos que utilicemos un esquema de redistribución dinámica de datos de acuerdo a los requerimientos actuales de los datos, esa situación ideal es difícilmente alcanzable. Es por ello que nuestro interés debe centrarse en encontrar la estrategia de particionamiento que provoque el menor desequilibrio de carga del sistema.

La figura 5-9 muestra cómo la estrategia de grano grueso es la que, de acuerdo a la discusión previa, peores resultados presenta. Más precisamente, en las consultas de tipo Q_4, es donde la estrategia de grano grueso muestra mayores desequilibrios. Por otra parte, Q_4 es el tipo más frecuente dentro de la Batería 3, y adicionalmente es el que captura el mayor grado de sesgo en la distribución de los datos. Por consiguiente, podemos concluir que la estrategia de grano grueso es la apuesta más arriesgada a la hora de elegir una estrategia de particionamiento multidimensional BEPI, si atendemos al importante criterio de rendimiento global del sistema.

Las dos estrategias restantes muestran un comportamiento muy similar respecto al equilibrio de carga por tipos de consulta. De hecho, todos los procesadores en ambas estrategias participan en la ejecución de todos los tipos de consulta. Además, el tiempo dedicado por un procesador al conjunto global de consultas es aproximadamente el mismo.

A pesar de las diferencias que se derivan de la figura 5-9, con el fin de tomar una decisión correcta es necesario estudiar el comportamiento de dichas estrategias cuando cambia el escenario de trabajo. Este estudio se presenta más adelante en este capítulo, cuando estudiemos el rendimiento de nuestras estrategias bajo criterios de escalabilidad y aceleración.

El modelo BEPI frente al modelo BCCT

Las distintas estrategias propuestas bajo el nombre de BEPI se enmarcan, como sabemos, en metodologías de particionamiento que no contemplan la posibilidad de utilizar un conocimiento previo de la carga de trabajo a la que se va a someter al sistema. Esta hipótesis de desconocimiento *a priori* de la utilización de la base de datos no es, sin embargo, de aplicación general en todas las situaciones. Resulta por ello interesante comprobar el rendimiento de una estrategia capaz de capturar el conocimiento anticipado de las necesidades de los usuarios y aplicaciones respecto a la frecuencia de accesos a la base de datos. Esta información puede además ser construida a partir de estadísticas que, durante un tiempo de utilización de la base de datos, vayan registrando los diferentes patrones de acceso a la base de datos.

Una vez conocidas las potencialidades de las estrategias de particionamiento BEPI basadas en árbol Q, en este apartado presentamos los resultados de un estudio experimental cuyo objetivo es el de poner de relevancia las mejoras que pueden conseguirse en la utilización de la estrategia BCCT en comparación con las estrategias BEPI.

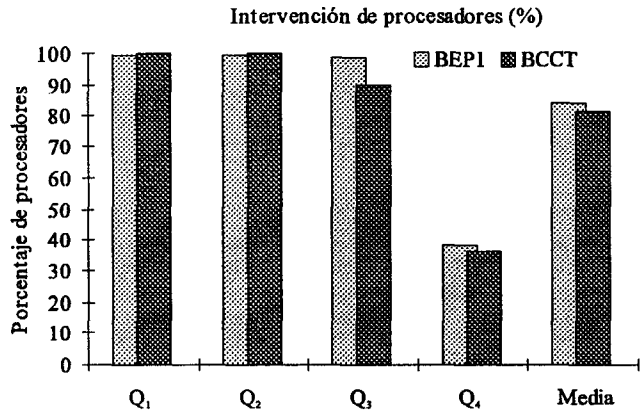


Figura 5-10. Comparación del porcentaje de procesadores que participan en la ejecución de consultas en los modelos BEPI y BCCT.

Antes de entrar a comentar los resultados obtenidos, conviene recordar que, en la aplicación del particionamiento BCCT y a partir del establecimiento de los requerimientos de una consulta típica, se calcula un valor óptimo para el número de procesadores *P* que deben participar en la ejecución de dicha consulta. La fórmula que permite obtener este valor de *P*, incluía un término que habíamos denominado *CP*, cuya presencia venía impuesta por el costo de participación de un procesador adicional en la ejecución de las consultas. Como ya hemos discutido con anterioridad, debido a que nuestro sistema no ha sido simulado en su totalidad, y por tanto no contempla tareas de control de concurrencia, tolerancia a fallos, planificación de procesos, etc., el establecimiento de un valor aproximado para *CP* resulta ciertamente arriesgado. Con el ánimo de evitar desajustes en los análisis comparativos de ambas alternativas de particionamiento, en nuestros experimentos hemos optado por considerar un valor de *CP* tal que hiciera óptimo un valor de ocho para el número de procesadores *P* que deben participar en la ejecución de una consulta típica. Esta simplificación nos permite entonces comparar estrechamente el comportamiento de las estrategias BEPI frente a la alternativa BCCT.

El primer estudio que debemos desarrollar en este ámbito consiste en comprobar si, en efecto, las consultas se ejecutan conforme a lo que establece BCCT. En este sentido, la figura 5-10 muestra que el número medio de procesadores implicados en la ejecución de una consulta media se sitúa por encima del 80% de procesadores en la máquina. Esto demuestra que la construcción a medida del árbol Q es correcta y que la fragmentación del espacio que realiza el mismo, se encuentra bien distribuida teniendo en cuenta que en nuestros planteamientos consideramos una máquina con exactamente ocho procesadores. Como puede notarse a partir de la figura 5-10, el número medio de procesadores utilizados en la ejecución de consultas es similar en ambas estrategias[3].

---

3 En la estrategia BEPI se ha utilizado una distribución de grano fino que, como vimos en los apartados previos, es la que mejores resultados proporciona a priori.

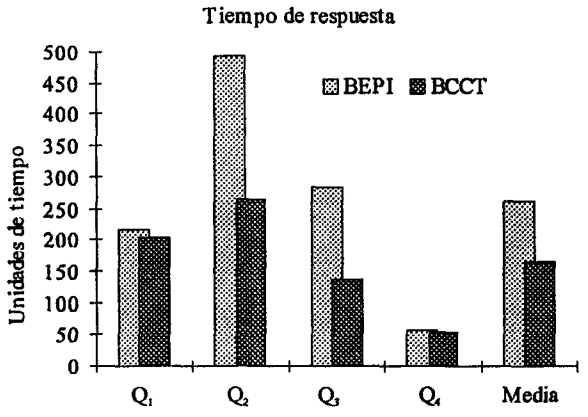


Figura 5-11. Comparación de los modelos BEPI y BCCT respecto a los tiempos de respuesta.

Si bien la figura anterior verifica el correcto planteamiento del particionamiento BCCT, tal resultado no es indicativo de ninguna mejora adicional respecto a la alternativa BEPI. La medición de tiempos de respuesta para ambas estrategias nos aporta ya algunas diferencias. Así, a partir de la figura 5-11 pueden extraerse las primeras conclusiones a este respecto. Como puede verse, los tiempos de respuesta obtenidos por BCCT son menores que para BEPI.

Dado que el árbol Q generado a partir de los cálculos efectuados por el enfoque BCCT, utiliza contenedores de cinco veces el tamaño de un contenedor convencional para el árbol Q de BEPI, en la ejecución de una misma consulta en ambos esquemas, con una alta probabilidad, el modelo BCCT procederá a examinar un número mayor de tuplas que el modelo BEPI. Sin embargo, en el primer caso las tuplas se distribuyen entre un menor número de grandes contenedores, contrariamente a como se almacenan en el segundo caso, donde se examina menor número de tuplas, pero repartidas entre un mayor número de contenedores más pequeños.

Teniendo en cuenta estas diferencias, los tiempos de acceso a contenedores en la estrategia BCCT serán, en general, menores que los tiempos de acceso en el modelo BEPI, ya que la lectura de un gran contenedor se hace secuencialmente, mientras que leer distintos contenedores del árbol Q de BEPI conlleva gastos adicionales de desplazamiento de cabezas ("*seek*") y latencia rotacional para cada contenedor. Así pues, las diferencias en tiempos de acceso a favor de la estrategia BCCT, pueden compensar el mayor número de tuplas que es preciso procesar con este modelo, en contraposición al modelo BEPI.

El razonamiento anterior podría inducirnos a pensar que, en cualquier caso, merece entonces definir contenedores de gran tamaño para el árbol Q. Sin embargo, observando los tiempos de respuesta mostrados por consultas de tipo Q_1 y Q_4, en las cuales el número de contenedores examinados es pequeño en comparación con otras consultas, podemos concluir que las diferencias no son importantes. En este caso, el tiempo invertido en transferir y procesar tuplas en grandes regiones se acerca mucho más al tiempo necesario para realizar accesos aleatorios a pocos contenedores de menor tamaño. De todo ello se deduce que la

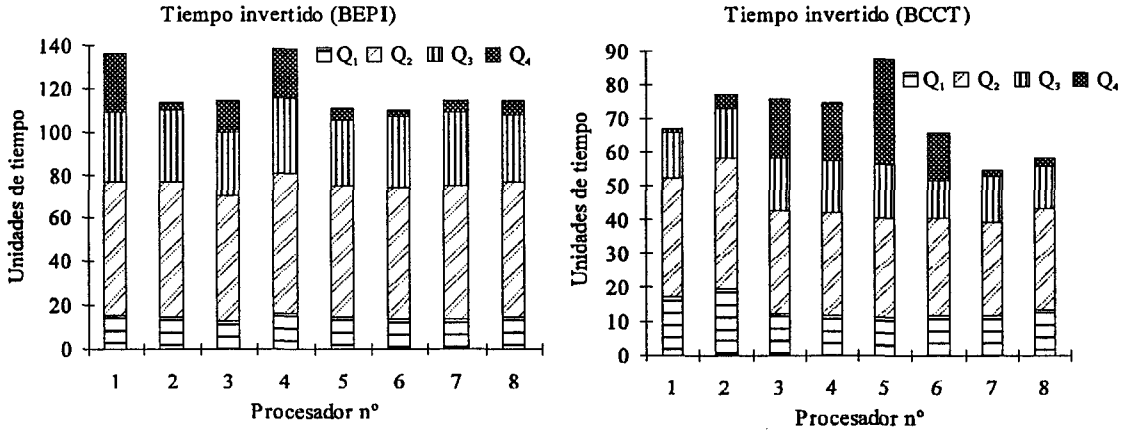


Figura 5-12. Tiempo invertido por procesadores individuales para los modelos BEPI y BCCT.

afirmación al comienzo de este párrafo no es necesariamente correcta. Para extraer conclusiones más concretas sobre este particular, sería necesario utilizar un modelo más preciso de dispositivo de disco, tal y como se describe en [74], que nos permitiera definir con mayor exactitud los costes de acceso a contenedores en función de los condicionantes típicos de este ámbito, tales como el tipo de disco utilizado, localización de los contenedores accedidos, tipo de bus utilizado, etc., así como un mecanismo definido de *buffering*. El establecimiento de un modelo de simulación de dispositivo de disco mucho más exacto que el utilizado en nuestras pruebas, es objeto de estudio actual por parte de nuestro grupo de trabajo.

Del mismo modo que en la comparación de los enfoques BEPI, nos centramos en el importante aspecto del equilibrio de carga por su influencia en el rendimiento global del sistema, es conveniente estudiar el comportamiento del particionamiento BCCT en este mismo sentido. Las gráficas en la figura 5-12 descubren el comportamiento de ambas estrategias respecto al equilibrio de carga. Como puede apreciarse, todos los procesadores participan en la ejecución de todos los tipos de consulta en ambos modelos de particionamiento. Como era de esperar, según vimos anteriormente, el tiempo de dedicación global de los procesadores a la batería completa de consultas es menor cuando se utiliza la estrategia de particionamiento BCCT. Sin embargo, las diferencias relativas en el tiempo invertido por los procesadores cuando se utiliza el modelo BCCT son más significativas que cuando se usa un modelo BEPI. En particular para las consultas de tipo Q_1 y Q_4, el modelo BCCT muestra un relativamente mayor desequilibrio del que presenta el modelo BEPI. Este mayor desequilibrio se debe a que con el árbol Q construido por el particionamiento BCCT, cuyos contenedores son de mayor tamaño que los del árbol Q utilizado por BEPI, la capacidad de filtrar tuplas es mucho menor, provocando así que algunos procesadores reciban un mayor número de tuplas pertenecientes a zonas calientes del espacio. La estrategia de grano fino utilizada para el particionamiento BEPI dispersa en mayor medida contenedores concentrados en zonas potencialmente calientes del espacio de búsqueda.

En un principio y a la vista de los resultados mostrados hasta el momento, se podría decir que, si bien el particionamiento BCCT consigue mejores tiempos de respuesta en una consulta típica de la Batería 3, desde el punto de vista del rendimiento global del sistema, el equilibrio que muestra la estrategia BCCT no supera los resultados obtenidos por BEPI considerando una distribución de grano fino.

En cualquier caso y debido al marco en el que nos movemos, ningún análisis sobre rendimiento de las estrategias de particionamiento estaría completo si eludimos la evaluación de tales estrategias respecto a criterios de aceleración y escalabilidad. El siguiente apartado refleja los aspectos más relevantes de este análisis.

3.2.3 Escalabilidad y aceleración

Un sistema paralelo ideal demuestra dos propiedades básicas: aceleración lineal y escalabilidad lineal. La primera de estas propiedades significa que doblando los recursos del sistema paralelo (doble número de nodos), el mismo trabajo puede efectuarse en la mitad de tiempo. La segunda de las propiedades indicaría que doblando el número de recursos del sistema, el doble de trabajo puede efectuarse en el mismo tiempo. Si bien estos objetivos son difícilmente alcanzables, es necesario tener un conocimiento aproximado de cómo se comportaría el sistema ante un incremento de los parámetros, tanto en número de recursos como en la propia carga a que se somete el sistema.

Para medir la influencia de los modelos de particionamiento en la capacidad de aceleración de un sistema paralelo, en el desarrollo de nuestros experimentos variamos el número de procesadores desde 8 hasta 32, mientras manteníamos constante el tamaño de la relación en 100.000 tuplas. Dado que las tuplas se generan siguiendo siempre los mismos patrones de distribución, los factores de selectividad de las consultas y por tanto su significado permanece prácticamente inalterable.

Ya que el árbol Q construido a medida para el particionamiento BCCT, utiliza el valor del número de procesadores óptimo (recordemos que era de ocho en nuestras pruebas) para definir el tamaño de los Q-nodos, en los experimentos previos, los contenedores gobernados por cada región se distribuían entre los ocho procesadores considerados. En este caso, al cambiar el número de procesadores, la forma de realizar la distribución en la estrategia BCCT consiste en asignar cíclicamente los contenedores gobernados por cada región entre los procesadores considerados. De este modo, si el último contenedor perteneciente a una región del árbol Q se asigna al procesador i, el primer contenedor de la siguiente región se asigna al procesador $(i+1)\ mod\ P$, siendo P el número total de procesadores de la máquina.

El primer resultado, relativo a las características de aceleración del sistema, se muestra en la figura 5-13a, donde se presentan los tiempos de respuesta de una consulta típica (valor promedio del tiempo de todas las consultas) de la Batería 3. Como era de esperar, los tiempos de respuesta disminuyen a medida que crece el número de nodos. Sin embargo, como puede apreciarse, la disminución más significativa se consigue cuando se utiliza el modelo de

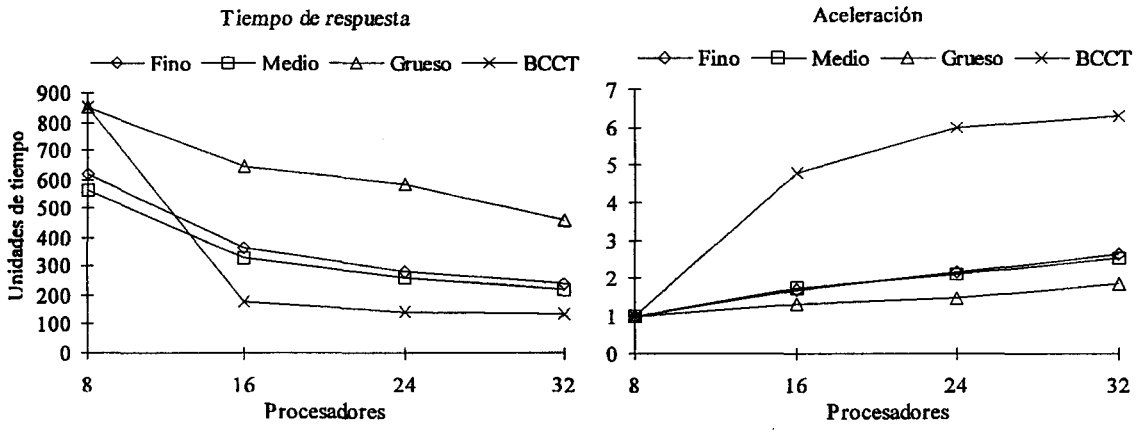


Figura 5-13. Propiedades de aceleración de los modelos BEPI y BCCT. (a) Tiempos de respuesta. (b) Curvas de aceleración correspondientes.

particionamiento BCCT. En este caso aparece además una reducción importante cuando pasamos de 8 a 16 nodos. Esta situación se deriva del hecho de que al haber doblado el tamaño de la relación original a 100.000 tuplas, los recursos necesarios para responder una consulta típica son también mayores y, en consecuencia, el valor de 8 como número óptimo de procesadores para ejecutar una consulta deja de ser válido. Sucede entonces que usando 8 procesadores para ejecutar una consulta, cada nodo debe ahora procesar un mayor número de grandes contenedores[4].

Las curvas de aceleración pertenecientes a estos tiempos de respuesta se muestran en la figura 5-13b. Esta segunda gráfica demuestra de forma más patente las buenas perspectivas que ofrece el modelo BCCT. Para esta última estrategia, la aceleración es superlineal. La razón de este comportamiento estriba en que a medida que se utilizan más nodos, los contenedores de gran tamaño se encuentran más repartidos. Esto supone una disminución en los tiempos de desplazamiento efectuados en cada nodo (con disco), lo cual compensa con mucho el coste de procesar un mayor número de tuplas que será preciso examinar en comparación con una estrategia BEPI, donde la reducción en número de desplazamientos no es tan acusada como en el modelo BCCT (debido al mayor número de contenedores de menor tamaño que es preciso procesar).

Por su parte, las estrategias de grano fino y grano medio ofrecen resultados razonables, aunque por debajo de un crecimiento lineal, como sería deseable. Esto demuestra que la distribución de contenedores en ambas estrategias mantiene un cierto equilibrio independientemente del número de procesadores entre los que se realice el reparto. Por supuesto, habrá un punto por encima del cual el costo de participación de un procesador adicional haga

4 En relación al tamaño de los contenedores utilizados en las estrategias BEPI.

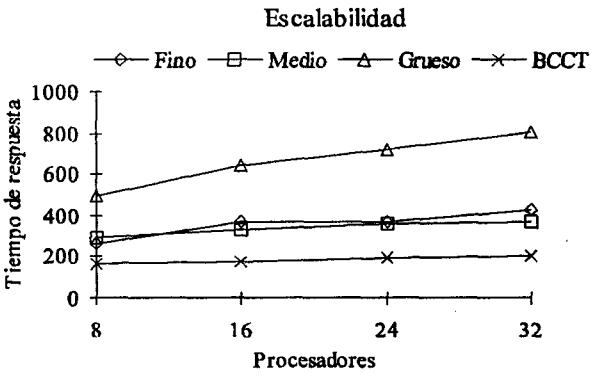


Figura 5-14. Figura de escalabilidad para las estrategias BEPI y BCCT.

decrecer la curva de aceleración. Este punto se alcanzará cuando el tiempo invertido por un procesador en ejecutar una tarea propia de una consulta sea menor que el costo de participación de un nuevo procesador.

El incremento sublineal mostrada por la aceleración en la estrategia BEPI de grano grueso, está provocado principalmente por la falta de equilibrio en la asignación de contenedores entre los procesadores a medida que crece su número.

Como experimento final, mostramos el rendimiento de las distintas estrategias de particionamiento basadas en árbol Q, ante criterios de escalabilidad. Para ello, en nuestras pruebas hemos incrementado el tamaño de la relación en el mismo orden en que aumentamos el número de procesadores de nuestra máquina simulada. Es decir, aumentamos progresivamente el tamaño de la relación desde 50.000 tuplas para los 8 procesadores hasta un total de 200.000 para la misma máquina con 32 procesadores.

La gráfica de la figura 5-14 presenta los tiempos de respuesta de una consulta típica de la Batería 3 para cada estrategia de particionamiento propuesta y bajo la variación del número de nodos de la máquina a medida que crece el tamaño del problema. La conclusión más importante que podemos extraer de dicha gráfica es que sus resultados verifican una vez más el correcto planteamiento de la estrategia BCCT. En efecto, puesto que el árbol Q configurado para esta estrategia se diseñó de acuerdo, por una parte, a los requerimientos de una consulta típica y por otra, al número de procesadores que deberían intervenir en la ejecución de dicha consulta, es de esperar que doblar los requemientos de recursos de una consulta al tiempo que se dobla el número de procesadores no afecte a los cálculos realizados. Sin embargo, este razonamiento sólo puede sostenerse siempre y cuando la distribución de carga de los nodos se mantenga en equilibrio. Por consiguiente el hecho de que el particionamiento BCCT presente una curva constante en los tiempos de ejecución para diferente número de nodos, demuestra una excepcional adaptabilidad del árbol Q a los sistemas paralelos de memoria distribuida.

Respecto a las estrategias BEPI, podemos extraer de la figura 5-14 que, de acuerdo a otros resultados mostrados con anterioridad, los enfoques de distribución de grano fino y grano medio se comportan de forma similar respecto a criterios de escalabilidad. Es destacable que, sin disponer de información previa sobre la carga de trabajo futura del sistema, ambas estrategias BEPI presenten una magnífica adaptación a la escalabilidad del sistema.

4 Conclusiones

En este capítulo se han descrito los aspectos más relevantes del trabajo experimental desarrollado como parte fundamental de la presente tesis. Si bien no ha sido nuestro objetivo exponer la forma en la que se han desarrollado los trabajos de implementación realizados, sí nos ha parecido más interesante (a la vez que revelador) presentar resultados obtenidos como consecuencia de los análisis efectuados sobre dichas implementaciones.

El trabajo experimental de la presente tesis ha tenido dos fases claramente diferenciadas. En un primer estado se ha llevado a cabo la implementación de la estructura de árbol Q como pieza clave del método de acceso para nuestra máquina QUATRO. La segunda fase en este apartado experimental ha consistido en la materialización de las estrategias de particionamiento multidimensional basadas en la anterior estructura.

Con objeto de encontrar puntos de referencia que nos permitiesen evaluar de algún modo la utilidad de nuestras propuestas, unido a todo lo anterior, un importante trabajo de análisis ha sido desarrollado en el marco de nuestra actividad experimental. Esta labor de análisis abarca la definición y generación de las distintas baterías de pruebas, la simulación del sistema sobre el cual hemos ejecutado dichas baterías y el posterior tratamiento de los resultados obtenidos.

Según los resultados mostrados acerca de la estructura de árbol Q, podemos destacar a modo de epílogo las siguientes conclusiones:

- Es un método de acceso multiatributo que permite indexar de manera eficiente tantos atributos de una relación como se deseen. Es decir, se trata de una estructura multidimensional **no sensitiva al número de dimensiones**.
- Consigue grados razonables de utilización del espacio disponible próximos a los que proporcionan estructuras básicas de indexación, tales como árboles B^+.
- Presenta muy **buenos niveles de discriminación** de datos, filtrando convenientemente un volumen de datos muy en concordancia con el grado de selectividad de la búsqueda.
- Muestra un **comportamiento aceptable** ante la presencia de un alto grado de **sesgo** en la distribución de los datos.

- Proporciona **reorganización dinámica**, tanto para la inserción de nuevas tuplas como en operaciones de borrado[5] de las mismas.
- Su adaptabilidad a un alto número de dimensiones convierte al árbol Q en un candidato idoneo para el **tratamiento de datos espaciales**.

En cuanto a la explotación de la estructura de árbol Q como soporte para el particionamiento multidimensional en sistemas paralelos de bases de datos, podemos resumir las características más destacables de nuestras propuestas en los siguientes puntos:

- El particionamiento multidimensional basado en árbol Q constituye una fórmula **original aplicable** de distribución de datos en máquinas de bases de datos paralelas.
- El modelo de particionamiento multidimensional muestra unos resultados **superiores** a los métodos actuales de **particionamiento lineal**, proporcionando menores tiempos de ejecución de consultas por varios atributos.
- La estructura de árbol Q proporciona la capacidad para distribuir de datos **basados o no** en el conocimiento previo de la carga futura de trabajo del sistema.
- Las estrategias de particionamiento multidimensional presentadas, exhiben favorecen el **equilibrio de carga** del sistema (respecto a la distribución de datos) en el tratamiento de conjuntos realistas de datos.
- Los modelos presentados de particionamiento basado en árbol Q son positivamente **sensibles a los parámetros de aceleración** del sistema.
- Las estrategias descritas ofrecen una excepcional **adaptación a la escalabilidad** del sistema.

Todos estos puntos recogen las características positivas más reseñables, tanto de la estructura propuesta como de las distintas metodologías de particionamiento multidimensional basadas en dicha estructura. Haciendo una seria reflexión al respecto, podemos decir que las conclusiones antes expuestas no son más que la afirmación de que nuestras propuestas presentan ciertas **potencialidades** deseables en el entorno para el que han sido concebidas.

Como es obvio, aunque se ha realizado un esfuerzo importante, primero en la implementación y posteriormente en la evaluación de las propuestas, aún queda trabajo por desarrollar. En concreto, no podremos conocer el comportamiento exacto de nuestras propuestas a menos que tengamos suficientemente desarrollado un modelo de simulación completo de nuestra máquina QUATRO que nos permita extrapolar de forma segura los resultados obtenidos. Hasta tanto esto sea posible, la conclusión más importante que,

5 Aunque en la actualidad los algoritmos de borrado para el árbol Q no están implementados, una primera versión de los mismos puede encontrarse en [5]. Dada la dimensión del problema, las operaciones de borrado constituyen objeto de implementación por otros componentes de nuestro grupo QUATRO.

particularmente, extraemos de todo lo expuesto, radica en que las enseñanzas obtenidas producto de este trabajo experimental nos permitirán afrontar con mayores garantías el futuro de nuestra investigación en el ámbito de las bases de datos paralelas.

Capítulo 6

Conclusiones y trabajos futuros

Una vez expuestos los planteamientos fundamentales que sustentan la tesis que presentamos, llega el momento de recapitular acerca del contenido global de la misma.

Aunque en cada capítulo de la presente memoria hemos expuesto, a modo de epílogo, las conclusiones más importantes referentes al tema sobre el que versa cada capítulo, es de rigor finalizar esta memoria de tesis doctoral, recogiendo las conclusiones que se derivan de todo lo anteriormente expuesto.

Como quiera que, a nuestro juicio, todo trabajo de tesis debe enmarcarse en una linea de investigación coherente, resulta preceptivo apuntar también las lineas por las que debe discurrir la investigación futura, con objeto de resolver aquellos problemas que han ido quedando abiertos en el desarrollo de este trabajo.

Con tales fines, en el presente capítulo se desarrollan sendos apartados destacando, por una parte, las conclusiones más relevantes del trabajo realizado y describiendo, por otra, las lineas de acción futuras que constituirán la continuación de la labor desarrollada durante estos últimos años.

1 Conclusiones

En el desarrollo de la presente memoria hemos descrito el método de indexación multiatributo en árbol Q, soporte de un conjunto de estrategias de particionamiento multidimensional de relaciones basadas en dicha estructura.

De la labor efectuada a lo largo del tiempo de preparación de la tesis cuya memoria concluimos en este capítulo, podemos extraer como principales conclusiones las que a continuación destacamos.

- Los modernos requerimientos, cada día más exigentes, de la mayoría de aplicaciones que, de uno u otro modo, precisan manejar volúmenes considerables de información, exigen nuevos mecanismos de acceso a esta información capaces de satisfacer con eficiencia las expectativas de sus usuarios. En tal sentido, esta tesis contribuye al desarrollo de nuevas técnicas de acceso a la información, con la aportación del método de acceso en árbol Q y de nuevos mecanismos de particionamiento multidimensional.
- La filosofía del paralelismo supone una vía de solución atractiva al problema del acceso eficiente a la información. El obstáculo que representan las operaciones de entrada/salida de datos en el tratamiento clásico de la información puede superarse con la aportación de nuevos mecanismos conducentes al acceso en paralelo de dicha información. El proyecto QUATRO, en cuyo seno se ha desarrollado la tesis que se presenta, representa nuestra propuesta global de solución en el ámbito del paralelismo a la problemática de acceso y gestión eficiente de la información.
- Del estudio realizado sobre las técnicas de particionamiento en los SGBDP, así como de los métodos de acceso a la información utilizados en tales sistemas, se desprende una cierta tendencia al conservadurismo en la utilización de mecanismos de almacenamiento y acceso a los datos. Esta resistencia al uso de técnicas más modernas de almacenamiento y acceso debe ser vencida con la inclusión de nuevos métodos de indexación cuyos rendimientos demuestren propiedades interesantes para su aplicación a tales sistemas.
- La estructura de árbol Q descrita en esta tesis representa un avance en el diseño de métodos de acceso multiatributo dinámicos y equilibrados, basados en el concepto de árbol paginado. Esta característica de paginación le confiere capacidades de almacenamiento y recuperación eficiente en dispositivos de disco.
- El trabajo efectuado de implementación del árbol Q nos permite aportar resultados muy aceptables respecto al rendimiento de la estructura. En cuanto al grado de utilización del espacio, la evaluación efectuada muestra rendimientos similares al de métodos de indexación de eficacia demostrada, tales como árboles B. Respecto a su eficiencia medida en número de accesos, los experimentos realizados exhiben unos resultados como mínimo prometedores.

- Dos mecanismos diferentes de particionamiento multidimensional basados en la explotación del árbol Q han sido desarrollados a lo largo de esta tesis. Desde una perspectiva genérica, el modelo multidimensional ofrece unos rendimientos superiores a métodos más convencionales de particionamiento lineal, basados en el uso de un sólo atributo. Desde una perspectiva más local, las comparaciones efectuadas de las distintas estrategias propuestas demuestran, por una parte, las propiedades excelentes del modelo BCCT respecto a criterios fundamentales en sistemas paralelos como son el equilibrio de carga, aceleración y escalabilidad y, por otra, el buen comportamiento de la sencilla estrategia de distribución de grano fino en el modelo BEPI.
- La inexistencia de baterías de pruebas ("*benchmarks*") específicamente diseñadas para la evaluación de métodos de acceso multiatributo y mecanismos de particionamiento, en las que se recoja la presencia de sesgo en los datos de entrada (muy frecuente en tipos de datos de uso habitual) no favorece la confrontación de nuevas propuestas frente a otras existentes. Esta circunstancia, nos ha empujado a invertir importantes esfuerzos en el desarrollo de baterías de pruebas específicas para medir diferentes aspectos del comportamiento de nuestras propuestas.

La experiencia acumulada durante el periodo de preparación de esta tesis nos permite asegurar que las propuestas que presentamos representan soluciones **viables** al problema del almacenamiento y acceso eficiente de la información en bases de datos paralelas. La fase experimental de esta tesis refrenda nuestra aseveración e induce a concebir fundadas esperanzas sobre la aplicabilidad y los potenciales beneficios de los mecanismos exhibidos a lo largo de esta memoria.

2 Lineas de trabajo futuro

Aunque en relación a cada uno de los temas que hemos ido tratando capítulo a capítulo, hemos identificado las diferentes vías de solución previstas para aquellos problemas que no son directamente abordados en esta tesis, en el presente apartado resumimos a grandes rasgos los trabajos que, incluidos en el ámbito en que se ubica esta tesis, tenemos previsto acometer en un futuro inmediato.

- Confección definitiva e implementación de los algoritmos de borrado en el árbol Q. En este sentido, como ya apuntamos en el capítulo 3, los trabajos se encuentran en una fase avanzada de desarrollo, ya que este problema fue originalmente emprendido en el marco del diseño global de la estructura de árbol Q. Sin embargo, las modificaciones que ha sufrido la estructura en el transcurso de nuestros trabajos, precisan de la revisión de los algoritmos originales de borrado para su establecimiento e incursión final en los programas que gestionan nuestra estructura.

- Desarrollo y análisis de los métodos de crecimiento dinámico del árbol Q en sistemas paralelos. Como ya apuntamos en el capítulo 4, el método DQG de crecimiento dinámico del árbol Q es objeto de dedicación actual por parte de nuestro grupo de trabajo. Esperamos a lo largo del presente curso obtener los primeros resultados que nos permitan aportar un mecanismo integral de almacenamiento distribuido y acceso paralelo a datos en SGBDP, fundamentado en el uso del árbol Q.
- Construcción de un modelo simulado de ejecución paralela de consultas lo suficientemente avanzado que nos permita extrapolar con garantías los resultados de las evaluaciones efectuadas. Los trabajos en esta dirección han comenzado con la programación de un refinado modelo de disco, que actualmente calibramos, para su aplicación inmediata como parte fundamental de la herramienta de evaluación de las distintas alternativas de particionamiento multidimensional. Este modelo, cuya descripción puede verse en [20], nos permitirá mejorar en breve los análisis efectuados sobre el rendimiento de las estrategias de particionamiento basadas en el árbol Q.
- Estudio de la viabilidad del árbol Q como estructura de datos multidimensional para el tratamiento de datos espaciales. La aparición de modernos sistemas de información geográfica, así como de aplicaciones que manejan datos espaciales, nos empuja a considerar esta vía de aplicación para nuestras estructuras. Las capacidades demostradas por la estructura del árbol Q, permiten acariciar fundadas expectativas de explotación en tales sistemas. Creemos que esta vía de explotación del árbol Q supone un interesante marco de aplicación con perspectivas que auguran resultados positivos en un tiempo no muy lejano.

Todas estas lineas de trabajo se enmarcan, como sabemos, dentro del proyecto QUATRO, cuyo desarrollo es el resorte que nos mueve -a cuantos participamos en él- para avanzar en las distintos areas de investigación en conexión con los sistemas de bases de datos paralelas.

No podemos terminar este apartado sin apuntar que, a pesar del empuje sufrido por la máquina QUATRO desde sus inicios, gran cantidad de trabajo sobre diversos aspectos de la máquina está aún por dedicar. Pero es esta circunstancia, unida a la actualidad y las atractivas perspectivas de futuro del proyecto QUATRO, la que posibilita ofrecer a nuevos compañeros un cauce de trabajo que aglutina los esfuerzos de todos nosotros, evitando los peligros de dispersión y desorientación propios de quien inicia labores de investigación y aportando, en definitiva, garantías de éxito en el desarrollo de esta labor.

Referencias

[1] Abdel-Ghaffar, K., El Abbadi, A., "Optimal Disk Allocation for Partial Match Queries", ACM TODS, Vol. 18, N° 1, Mar. 1993.

[2] Apers, P., Berg, C., Flokstra, J., Grefen, P., Kersten, M. Wilschut, A., "PRISMA/db: A Parallel, Main Memory Relational DBMS" IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 6, Dec. 1992.

[3] Barrena, M., De Miguel, P., Hernández, J., Martínez, J., Polo, A., Nieto, M., "QUATRO: A parallel database management model". Technical report N° FIM66.1/ARQ/92. Technical University, Madrid 1992.

[4] Barrena, M., De Miguel, P., Hernández, J., Martínez, J., Polo, A., Nieto, M., "Parallel Transaction execution in the QUATRO parallel database machine", Parallel Computing and Transputer Applications, Part II, eds: M. Valero, E. Oñate, M. Jane, J.L. Larriba y B. Suárez, IOS Press, Septiembre 1992.

[5] Barrena, M., De Miguel, P., Hernández, J., Martínez, J., Polo, A., Nieto, M., "The Q-tree: A Multiattributo search structure". Technical Report n° FIM/79.1/datsi/94, Mayo 1994.

[6] Barrena, M., De Miguel, P., Hernández, J., Martínez, J., Polo, A., Nieto, M., "The Q-tree: A Balanced Multiattribute Indexing Method for Database Management Systems", Informe técnico interno. Dpto Informática. Universidad de Extremadura. 1994.

[7] Bayer, E., McCreight, E., "Organization and Maintenance of Large Ordered Indexes", Acta Informática, Vol. 1, 1972.

[8] Bentley, J.L. "Multi-dimensional binary search trees used for associative searching". Commun. ACM. Vol 18, N° 9. Sept. 1975.

[9] Bentley, J.L. and W.A. Bukhard, "Heuristics for partial match retrieval data base design", Inform. Process. Lett., vol 4, pp. 132-135, Feb. 1976

[10] Bentley, J.L., "Multidimensional Binary Search Trees in Database Applications", IEEE Trans. on Software Engineering, Vol. SE-5, No. 4, 1979.

[11] Bergsten, B., Couprie, M., Valduriez, P., "Prototyping DBS3, a Shared-Memory Parallel Database System". Int. Conf. on Parallel and Distributed Information Systems, Miami, 1991.

[12] Bhide, A., "An Analysis of Three Transaction Processing Architectures", Proc. 14th VLDB Conf. 1988. pp. 339-350

[13] Boral, H.et al, "Prototyping Bubba, A Highly Parallel Database Sistem", IEEE Trans. on Knowledge and Data Engineering, pp: 4-24, vol 2(1), March 1990.

[14] G. Bültzingsloewen et al., "Design and Implementation of KARDAMON - A Set-Oriented Data Flow Database Machine", Proceedings of the Sixth International Workshop on Database Machines IWDM'89, pp: 18-33, H. Boral and P. Faudemay (Eds), LNCS 368, Springer-Verlag 1989.

[15] Cariño, F., Sterling, W., Kostamaa, P., "Exegesis of DBC/1012 and P-90—Industrial Supercomputer Database Machines", Proc. of the Fourth Int. PARLE Conf., Paris. Springer-Verlag, June 1992.

[16] Copeland, G., Alexander, W., Boughter, E., Keller, T., "Data Placement In Bubba". Proc. of ACM-SIGMOD Int. Conf. on Management of Data. Chicago, May, 1988.

[17] DeWitt, D.J. et al., "The GAMMA Database Machine project", IEEE Trans. on Knowledge and Data Engineering, pp: 44-62, vol 2(1), March 1990.

[18] DeWitt, D.J. et al. "Practical Skew Handling in Parallel Joins", Proc. 18th Int'l Conf. Very Large Data Bases, Morgan Kauffman, San Mateo, Calif., 1992.

[19] DeWitt, D.J., and Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", Comm. of the ACM, Vol 35, No. 6, June 1992, pp. 85-98.

[20] Díaz, P., Barrena, M. "Modelo de simulación de disco". Documento interno. Dpto de Informática. Universidad de Extremadura. Cáceres, 1995.

[21] Du, H., Sobolewski, J., "Disk Allocation for Cartesian Product Files on Multiple-disk Systems", ACM TODS, Vol. 17, N° 1, Mar 1982.

[22] Evangelidis, G., Lomet, D., Salzberg, B., "The hB^{Π}-tree: A Concurrent and Recoverable Multi-attribute Index Structure", Tesis Doctoral, Northeastern University, Junio 1994.

[23] Finkel, R., Bentley, A., "Quad tress: a data structure for retrieval on composite keys", Acta Informatica 4, 1974.

[24] Freeston, M., "The BANG file: a new kind of grid file", ACM, 1987.

[25] Frieder, O., Mason, G., "Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems", IEEE Computer, Nov. 1990.

[26] Friedman, J.J., J.L. Bentley and R.A. Finkel, "An algorithm for finding best matches in logarithmic expected time", ACM Trans. Mathematical Software, vol. 3, pp. 209-226. Sept 1977.

[27] Ghandeharizadeh, S., DeWitt, D.J., "MAGIC: A Multiattribute Declustering Mechanism for Multiprocessor Database Machines", IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 2, May 1994.

[28] Graefe, G., "Encapsulation of Parallelism in the Volcano Query Processing System", ACM SIGMOD Conf., May 1990

[29] Gray, J. "The Benchmark Handbook for database and transaction processing systems", second edition. Morgan Kauffman Publishers Inc. 1993.

[30] Gray, J., Reuter, A. "Transaction processing: concepts and techniques", Morgan Kaufmann Publishers, 1993.

[31] Güting, H., Kriegel, H., "Multidimensional B-tree: An efficient dynamic file structure for exact match queries", Proc. 10th GI Annual Conference, Informatik Fechberichte Band 33, Springer-Verlag, 1980.

[32] Guttman, A. "R-trees: a dinamic index structure for spatial searching". Proc. ACM SIGMOD Conf., Boston 1984. pp. 47-57.

[33] Hernández, J., "Modelo de programación basado en actores para sistemas masivamente paralelos". Tesis Doctoral. Facultad de Informática. Universidad Politécnica de Madrid. 1995.

[34] Hua, K., Lee, C., "An adaptive data placement scheme for parallel database computer systems", Proc. Int. Conf. VLDB 90.

[35] Hua,K., Lee, C., "Handling Data Skew in Multiprocessor Database Computers Using Partitioning Tuning", Proc. 17th Int'l Conf. Very Large Data Bases, Morgan Kauffman, San Mateo, Calif., 1991.

[36] Hutflesz, A., Six, H., Widmayer, P., "Twin grid files: Space optimizing access schemes", Proc. of ACM SIGMOD Conference, New York, 1988.

[37] Kelley, K., Rusinkiewicz, M., "Multikey, extensible hashing for relational databases", IEEE Software, July 1988.

[38] Kim, M., Pramanik, S., "Optimal File Distribution for Partial Match Retrieval", Proc. ACM SIGMOD, May 1988.

[39] Krigel, H., Vaishnai, V., "Weighted multidimensional B-trees used as nearly optimal dynamic directories", Proc MFCS'81, Lecture Notes in Computer Science, No. 118, Springer-Verlag, 1981.

[40] Kronenberg, N., Levy, H., Strecker, W. "VAXclusters: A Closely-Coupled Distributed Systems". ACM Trans. on Computer Systems, Vol 4, No. 2. May 1986.

[41] Lakshmi M.S.,Yu, P.S., "Effectiveness of Parallel Joins", IEEE Trans. Knowledge and Data Engineering, Vol. 2, No. 4, Dec. 1990, pp. 410-424.

[42] Li, J., Srivastara, J., Rotem, D., "CMD: A Multidimensional Declustering Method for Parallel Database Systems", Proc. VLBD Conf., 1992.

[43] Lomet, D. B., "A simple bounded disorder file organization with good performance". ACM Trans. Database Systems, Vol. 13, No. 4. Dec. 1990.

[44] Lomet D. B. , Salzberg B. "The hB-Tree: A Multi-attribute Indexing Method with Good Guaranteed Performance". ACM Trans. on Database Systems, Vol. 15, N° 4, December 1990

[45] Lomet, D. "A Review of Recent Work on Multi-atribute Access Methods", SIGMOD RECORD, Vol 21, No. 3, Sep, 1992.

[46] Lorie, R.A., Daudenarde, J. J., Stamos, J. W., Young, H.C.,"Exploiting Database Parallelism in a Message-Passing Multiprocessor", IBM Journal of Research and Developments 35, No. 5/6, 681-696. 1991.

[47] Lu, H., Ooi B-C., Tan, K-L., "Query Processing in Parallel Relational Database Systems". IEEE Computer Society Press, Los Alamitos, Calif. 1994.

[48] Macias J.A., Barrena M., "DQG: Un método de distribución dinámica del árbol Q en sistemas paralelos", Documento interno. Dpto. Informática. Universidad de Extremadura. Cáceres, 1995.

[49] Missikoff, M. et al., DBMAC: A Parallel Relational Database Machine, in Database Machines: Modern Trends and Applications, pp: 85-126, Edited by A. Sood and A. Qureshi, NATO ASI Series - Springer Verlang, 1986.

[50] Mohan, C., Pirahesh, H., Tang W.G., Wang, Y., "Parallelism in relational database management systems" IBM Systems Journal, Vol 33, No. 2, 1994.

[51] Montgomery, A.Y., D'Souza, D.J., Lee, S.B., "The Cost of Relational Algebraic Operations on Skew Data: Estimates and Experiments", Information Processing 83, 1983.

[52] Nievergelt, Hinterberger, and Sevcik. "The grid file: An adaptable, symmetric multikey file structure". ACM Trans. Database Syst. Vol. 9, N° 1, March 1984.

[53] Orenstein, J., Merret, T., "A class of data structures for associative searching". Proc. ACM PODS Conf., Waterloo, 1984. pp. 181-190.

[54] Otoo, E., "A Multidimensional Digital Hashing Scheme for Files with Composite Keys". ACM 1985.

[55] Ouksel, M., Sheuerman, P., "Multidimensional B-trees: analysis of dynamic behaviour", BIT 21, 1981.

[56] Ouksel, M., Scheuermann, P., "Storage Mapping for Multidimensional Linear Dynamic Hashing". Proc. of 2nd Symposium on Principles of Database Systems. 1983.

[57] Ozkarahan, E., Ouksel, M., "Dynamic and order preserving data partitioning for database machines", Proc. Int. Conf. VLDB, 1985.

[58] Patterson, D.A., Gibson, G. and Katz, R. H., "A case for redundant arrays of inexpensive disks (RAID)". In Proceedings of the ACM-SIGMOD International Conferencie on Management of Data. Chicago, May 1988.

[59] Piradesh, H., Mohan, C., Cheng, J., Liu, T., Selinger, P., "Parallelism in Relational DAta BAse Systems: Architectural Issues and Desing Approaches", IEEE 1990.

[60] Polo A., Barrena M., De Miguel P., Hernández J., Martínez J.M., Nieto M., "Multidimensional Partitioning for Masivelly Parallel Database Systems".Proc. 3th Euromicro Workshop on Parallel and Distributed Processing. IEEE computer Society Press. Jan. 1995.

[61] Richardson, J.P., Lu, H., Mikkilineni, K., "Desing and Evaluation of Parallel Pipelined Join Algorithms", Proc. ACM SIGMOD, 1987, pp. 399-409.

[62] Robinson, J. "The K-D-B-tree: a search structure for large multi-dimensional dynamic indexes", Proc. ACM SIGMOD Conf., April 1981,

[63] Rotem, D., Segev, A., "Algorithms for Multidimensional Partitioning of Static Files". IEEE Trans. on Software Engineering, Vol 14, No. 11, Nov. 1988.

[64] Rotem, D. "Clustered multiattribute hash files", 8th Symposium on Principles of Database Systems. 1989

[65] Scrutchin, T. "TPF: Performance, Capacity, Availability", Proc. IEEE Compcon Spring '87. San Francisco, Feb. 1987.

[66] Sekino, A. et al., "The DCS - A New Approach to Multisystem Data Sharing", Proc. Nat. Computer Conf., AFIPS Press, Reston, Va., 1984, pp. 59-68.

[67] Sellis, T., Roussopoulos, N., Faloutsos, C., "The R+-tree: a dynamic index for multi-dimensional objects". Proc. VLDB Conf., Bringhton, 1987.

[68] M. Stonebraker, D. Patterson, and J. Ousterhout, "The design of XPRS", Proc. Int. Conf. Very Large Databases, Los Angeles, Sept. 1988.

[69] Strickland, J.P., Uhrowczik, P.P., Watts,V.L., "IMS/VS: An Evolving System", IBM Systems Journal 21, No. 4, 490-510, 1982.

[70] Tanaka, Y., "Adaptive Segmentation Schemes for Large Relational Database Machines", Database Machines Int. Workshop, Springer-Verlag, Munich, 1983.

[71] The Tandem Performance Group, "A Benchmark of NonStop SQL on the Debit Credit Transaction", Proceedings of the SIGMOD 88 Conference, pp: 337-341, ACM SIGMOD Record vol 17(3) September 1988.

[72] Valduriez, P., "Parallel Database Systems: Open Problems and New Issues", Distributed and Parallel Databases - An International Journal, Vol. 1(2), April 1993. Kluwer Academic Publishers.

[73] Walton, C.B., Dale, A.G., Jenevein, R.M., "A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins", Proc. 17th Int'l Conf. on Very Large Data Bases, Morgan Kaufman, San Mateo, Calif., 1991.

[74] Wilkes, J., Ruemmler, C., "An Introduction to Disk Drive Modeling". IEEE Computer. Vol. 27, No. 3, March, 1994

[75] Wolf, J.L. et al. "An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew", Proc. 7th Int'l Conf.Data Eng., IEEE CS Press, Los Alamitos, Calif., 1991.